Bootstrap 4

Conheça a biblioteca front-end
mais utilizada do mundo

Casa do Código

SÉRIE CAELUM

NATAN SOUZA

ISBN

- Impresso e PDF: 978-85-94188-60-1
- EPUB: 978-85-94188-61-8
- MOBI: 978-85-94188-62-5

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

AGRADECIMENTOS

"Heroes never die" — Mercy, Overwatch

Lembro que minha primeira aula teste na Caelum foi bem tensa. O Alberto, que tocava essa parte do treinamento na época, falou para escolher qualquer tema que eu dominava. Então, escolhi Bootstrap sem pensar. Péssimo primeiro teste, ótima primeira escolha.

O feito de escrever um livro sempre é algo desafiador, mas a sensação boa de recompensa depois de um árduo trabalho é reconfortante, lembrando que não o escrevi sozinho.

Quero agradecer à Caelum e a todos os "caelumers" por me ajudarem direta, ou indiretamente, na concepção deste livro, desde um papo motivador até as revisões técnicas e gramaticais. Agradeço também a todos os meus alunos, que sempre me instigam a estudar mais e mais.

E muito obrigado a você, leitor, que investiu seu dinheiro neste livro bem como investirá seu tempo também.

Dedico-o a Paula Midori, Claudia e Ivo, pelas enormes doses de paciência, amor e conselhos que me dão diariamente. São meus três fortes pilares que tenho na vida e tenho certeza de que, sem eles, não seria nada.

SOBRE O AUTOR

Natan Souza é front-end designer no grupo Caelum desde 2015, e instrutor dos cursos presenciais de front-end e UX. Além disso, também produz cursos online dessas áreas para a Alura, incluindo os de UX, acessibilidade e pré-processadores.

Começou a dar seus primeiros cliques no Photoshop ainda em 2005, o que o levaria a se interessar pela área de Design, e graduando-se bacharel em Design Digital anos mais tarde. Está focado na área de web e UX desde 2009, passando por empresas como FIAP e PMESP. Atuou como front-end e designer em toda a sua trajetória profissional.

- Twitter: *@designernatan*
- LinkedIn: http://bit.ly/linkedinDoNatan
- Site: http://www.designernatan.com.br

BOOTSTRAP, A ORIGEM

Imagine que você faz parte da equipe de front-end da Juquinha Consultoria, uma importante consultoria de TI. Você e seu time desenvolvem um guia de estilos com CSS que logo se torna padrão na empresa inteira, inclusive na sede.

Passado um tempo, vocês se perguntam se outras equipes ao redor do mundo não se beneficiarão do uso deste guia, e resolvem disponibilizá-lo online, no GitHub, totalmente *open source*. O repositório/projeto é um sucesso, e cai no gosto da comunidade mundial de desenvolvimento, principalmente entre os desenvolvedores back-end – e talvez também seja um pouco odiado pelos designers (veremos o motivo disso adiante).

Esta história é baseada em fatos reais, e aconteceu com uma "pequena" empresa chamada Twitter que, em agosto de 2011, lançou para o mundo seu guia de estilos em CSS, que ficaria famoso por muitos anos, o **Bootstrap**. É isso mesmo, o Bootstrap é, resumidamente, um grande arquivo CSS com uma excelente documentação (diga-se de passagem) e que possui dezenas e dezenas de componentes prontos.

Fazer um site elegante com três colunas nunca foi tão fácil, mesmo para quem nem sabe escrever uma linha de CSS e, muito menos, algo sobre harmonia das cores. Foi aí que começaram as divergências, pois, se de um lado temos desenvolvedores back-end felizes em não precisar mexer com CSS e/ou design, tínhamos *front-enders* bravos, pois o HTML ficava muito sujo, e os designers mais bravos ainda, já que a cara de todo site feito em Bootstrap

ficava igual!

Falaremos mais disso em capítulos posteriores, mas o fato é que a criação do Bootstrap foi um marco para a área de desenvolvimento. Sistemas internos no geral começaram a ter um trabalho mais bem finalizado com relação a estrutura e layout por conta dele.

Herói ou vilão? Batman ou Coringa? Você vai conferir neste livro como trabalhar com essa fantástica biblioteca em sua última versão 4, e verá alguns de seus segredos mais profundos, algumas de suas fraquezas e, principalmente, suas características mais notáveis.

O livro é focado em profissionais ligados à tecnologia, que possuam conhecimentos básicos de HTML/CSS e que **nunca tiveram contato com o Bootstrap**.

Se você já conhece o Bootstrap 3 e quer apenas se atualizar, esta leitura não é indicada. Mas recomendo dar uma olhada no próprio blog deles: http://blog.getbootstrap.com/2018/01/18/bootstrap-4/.

Agora, vamos começar a brincar de Bootstrap!

Sumário

1 O projeto Better **1**

1.1 E no mobile? 15

1.2 Terminando o menu 22

1.3 Resumo 29

2 Destacando o que deve ser destacado **31**

2.1 Relembrando o wireframe e colocando a mão na massa 32

2.2 Jumbo quem? 35

2.3 Tirando as margens 41

2.4 Estilizando um campo de formulário 48

2.5 Campo e botão lado a lado 52

2.6 Resumo 57

3 Entendendo as grids do Bootstrap **59**

3.1 Analisando o wireframe da galeria 60

3.2 Começando a marcação 62

3.3 Centralizando tudo 66

3.4 Melhorando a galeria 68

3.5 Um site responsivo para telas maiores 71

3.6 Ajustando para telas maiores 78
3.7 Resumo 84

4 Flexbox, agora no Bootstrap 86
4.1 Não é TV Fama, é Hall da Fama! 89
4.2 O flexbox mais rápido do oeste 95
4.3 Ajeitando o hall 98
4.4 Resumo 104

5 Media, um componente atualizado 106
5.1 Um antigo componente 107
5.2 Resumo 122

6 Cartas na mesa: brincando com o card 124
6.1 Começando com o plano 125
6.2 Estilizando nosso primeiro card 128
6.3 Aplicando a grid aos cards 135
6.4 Mudando o estilo do plano principal 137
6.5 Fazendo o rodapé 139
6.6 Resumo 148

7 Classes semânticas e CSS mais leve 150
7.1 Diminuindo a poluição no HTML usando Sass 150
7.2 Preparando a infra 152
7.3 Juntando regras CSS em uma só classe 154
7.4 Escolhendo o que usar do Bootstrap 161
7.5 Resumo 167

8 Conselhos e dicas 168
8.1 Qual é o melhor framework CSS? 169

8.2 Links de saideira 170

Capítulo 1

O PROJETO BETTER

Neste livro, criaremos juntos um projeto do começo ao fim usando o Bootstrap 4 e seus principais componentes, como menu responsivo, *grids*, painéis etc. Além de customizar o tema padrão, usaremos boas práticas de HTML5 levando em consideração a semântica do nosso código e teremos a performance do projeto como um ponto a ser pensado.

Nosso cliente, Vasco Nakombi, possui um projeto chamado Better, um banco de imagens pago, no qual o usuário pode baixar determinadas fotos de graça, mas precisa pagar por outras. No *briefing* minimalista passado por Vasco, foi determinado que "rápido e responsivo" devem ser os pilares para toda decisão que envolva o desenvolvimento da plataforma, logo, já foi descartado fazer pesquisa de público-alvo ou mesmo qualquer dinâmica de UX.

Além disso, o *wireframe* já foi aprovado e devemos segui-lo fielmente. Precisamos de algo que seja rápido e fácil de usar.

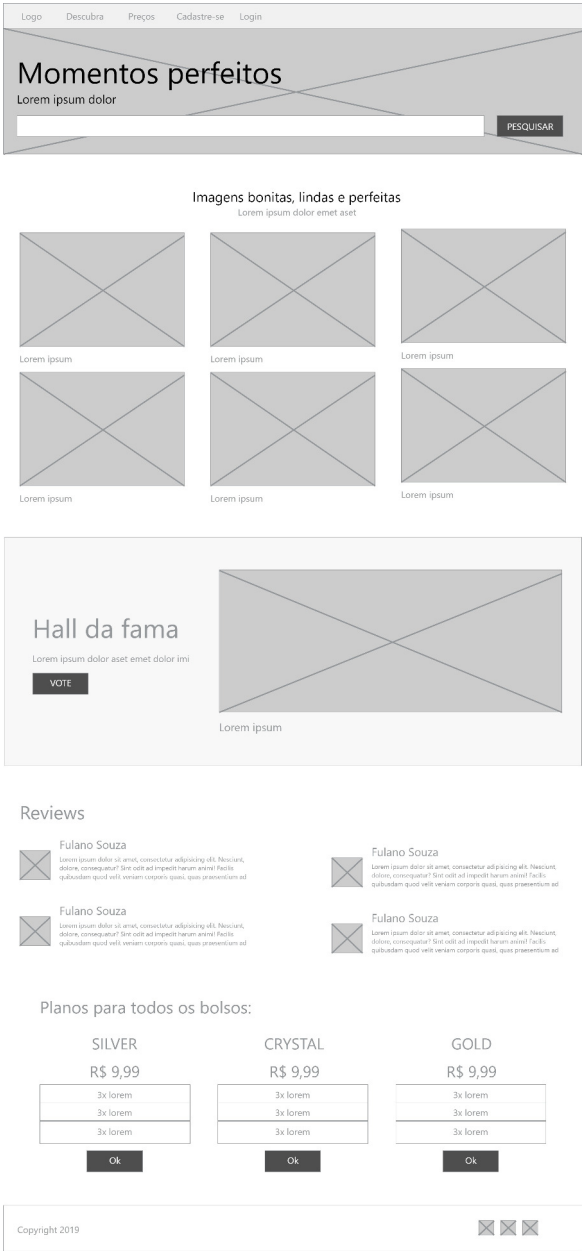


Figura 1.1: Wireframe Home Better

Você pode acessar esse wireframe nesse link: https://github.com/designernatan/livro-bootstrap4/blob/master/wireframe-home.png

Além da home, por solicitação do cliente, precisaremos criar a página de cadastro e a página de login. A Better ainda não possui um site funcional com HTML/CSS, e o designer responsável por fazer a parte da criação foi demitido. Como podemos proceder, sendo que, na equipe, temos apenas programadores *back-end* com conhecimento básico de front-end? E quanto ao design? Talvez, a saída seja fazer um curso online na Alura, ou ler um livro de front-end da Casa do Código.

Porém, qual era um dos itens primordiais para o cliente? Tempo! Assim, não temos tempo nem *budget* para instruir os programadores. Eis que surge a cavalaria: o Bootstrap.

Como dito na introdução deste livro, o Bootstrap é um *framework* CSS repleto de componentes prontos e reutilizáveis. Podemos usá-lo para ganhar tempo e deixar os programadores focados em programar de fato.

Framework ou biblioteca?

Primeira curiosidade: em suas versões alpha e beta, o Bootstrap 4 considerava-se um framework. Já na sua versão final, o termo escolhido mudou para biblioteca.

Há diversas discussões na internet com argumentos realmente fascinantes, mas não me prenderei muito a terminologias aqui, pois, se o próprio Bootstrap mudou a forma de se chamar, quem sou eu para argumentar algo contra?

Primeiramente, teremos de preparar nosso ambiente de desenvolvimento. Acredito que você, leitor, provavelmente possua uma IDE preferida; eu usarei o Sublime Text, mas sinta-se à vontade para usar o seu editor de texto do coração, seja Brackets, Atom, Visual Studio Code, VIM etc.

No link a seguir, baixe os arquivos do projeto (30 MB) que usaremos no decorrer do seu desenvolvimento e descompacte todo o seu conteúdo em alguma pasta, seja no Desktop ou em uma pasta do seu Dropbox, por exemplo: https://github.com/designernatan/livro-bootstrap4.

Outras pastas

No mesmo repositório, você poderá encontrar os projetos finais de cada capítulo, devidamente identificados.

Agora pegue a pasta `projeto-better` e abra-a em seu editor de texto. Depois, abra o arquivo `index.html` para começarmos a editar seu código. Perceba que o que já consta no arquivo é a estrutura básica de um HTML:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width">
    <title>Better - Fotos de graça e pagas</title>

  </head>
  <body>

  </body>
</html>
```

> **Onde baixo o Bootstrap?**
>
> Se você quiser baixar o Bootstrap 4 (e até versões anteriores), basta acessar seu site principal, que também contém toda a sua documentação: https://getbootstrap.com/.

Quando vamos fazer uma redação, normalmente começamos, bem... Do começo. Logo, faremos o mesmo com o projeto, começando do topo. Vamos partir da estrutura básica do menu de navegação:

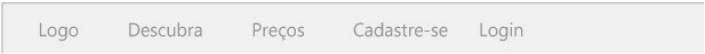


Figura 1.2: Wireframe do menu de navegação

Um menu de navegação desse tipo consiste basicamente em

uma lista não ordenada com vários itens de lista, sendo cada um deles um link. Vamos fazer o código dentro do `<body>` desta forma:

```
<ul>
  <li><a href="#">Better</a></li>
  <li><a href="#">Descubra</a></li>
  <li><a href="#">Preços</a></li>
  <li><a href="#">Cadastre-se</a></li>
  <li><a href="#">Login</a></li>
</ul>
```

Não se preocupe com as URLs dos links, vamos alterá-las em capítulos posteriores. Abrindo a página no browser, não teremos nada muito bonito:

Figura 1.3: Menu sem estilo

Poderíamos criar regras CSS para estilizar esse menu e colocar um `display: inline` nessas `<li>` . Mas, já que vamos utilizar o Bootstrap aqui, por que se preocupar? Como dito anteriormente, ele é basicamente um arquivo CSS, então, vamos importá-lo para ver como fica. Para importar um arquivo CSS, basta colocarmos a tag `<link>` desta forma:

```
<link rel="stylesheet" href="/css/bootstrap.css">
```

Feito isso, abrindo no browser:

Figura 1.4: Menu com Bootstrap importado

Repare que, mesmo sem mudar nenhum elemento de estrutura nem colocar nenhum tipo de classe, o resultado final ficou diferente. Os links estão com um tom de azul mais tênue, e até mesmo a tipografia foi alterada, além dos espaçamentos que foram modificados.

Mas como aproximar ainda mais do resultado esperado? Como deixar o menu na horizontal conforme o *wireframe* passado usando o Bootstrap? Talvez, possamos usar um elemento do HTML5 que traga mais semântica? Vamos tentar isso com o elemento `<nav>` :

```
<nav>
  <ul>
    <li><a href="#">Better</a></li>
    <li><a href="#">Descubra</a></li>
    <li><a href="#">Preços</a></li>
    <li><a href="#">Cadastre-se</a></li>
    <li><a href="#">Login</a></li>
  </ul>
</nav>
```

Testando no browser, nada muda. Isso se deve ao fato de que o `<nav>` não influencia realmente em nada do layout, e sim acaba apenas trazendo mais significado para o nosso código, **mais semântica**. Perceba que a semântica do código depende unicamente do desenvolvedor que está *codando*, e não do

framework. Agora como será que o Bootstrap trata menus?

Esse fantástico framework possui classes específicas para fazer menu de navegação, como também componentes prontos que nos auxiliam nessa tarefa. Um desses componentes é o chamado *navbar*. Mas como utilizá-lo? Precisaremos aprender alguma linguagem back-end, como Java ou PHP?

Nada disso! Vamos relembrar que o Bootstrap é basicamente um CSS, logo, para usarmos seus componentes prontos, basta sabermos os nomes das classes que ele possui e aplicá-las em nosso HTML.

Como saber o nome dessas classes? Basta abrir o arquivo `bootstrap.css`, localizado na pasta `css`, e procurar nas mais de **seis mil linhas de código**. Será que não existe algum tipo manual do Bootstrap, por exemplo, algum local no qual existam exemplos e um jeito melhor de achar os componentes? Sim, existe, basta apenas acessar seu site, https://getbootstrap.com.

Nele, podemos ter acesso a toda a documentação detalhada do framework. Ela é extremamente bem cuidada e é um dos outros motivos por que o Bootstrap ficou tão famoso:

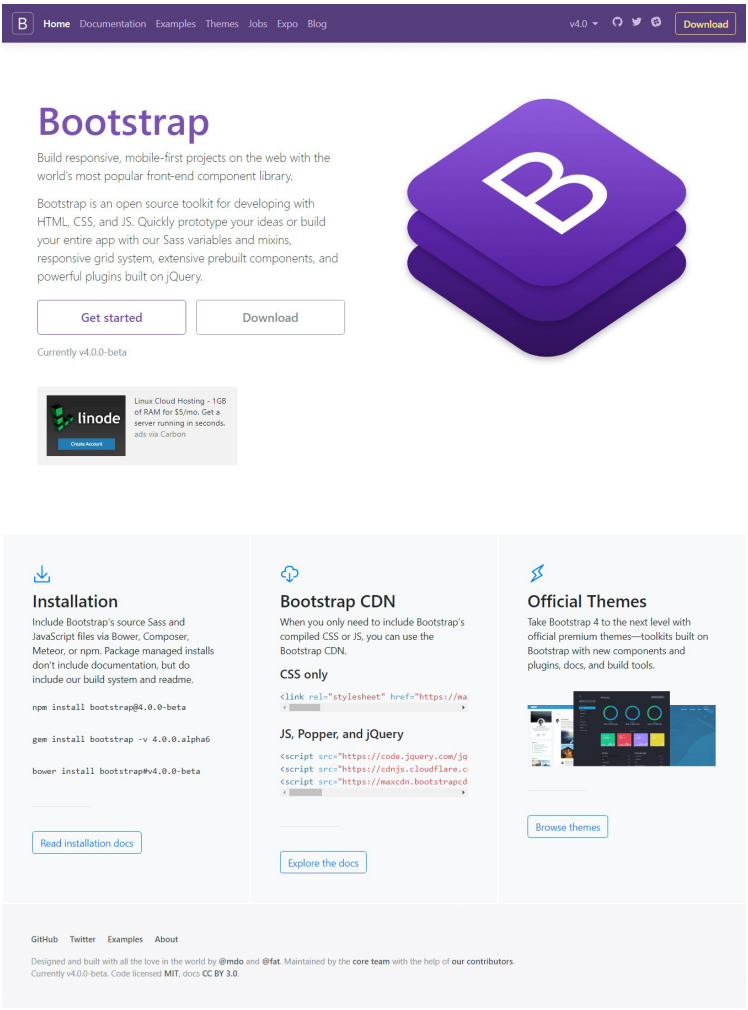

Home Documentation Examples Themes Jobs Expo Blog
v4.0
Download
Bootstrap
Build responsive, mobile-first projects on the web with the world's most popular front-end component library.
Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.
Get started
Download
Currently v4.0.0-beta
linode
Linux Cloud Hosting - 1GB of RAM for $5/mo. Get a server running in seconds.
ads via Carbon
Installation
Include Bootstrap's source Sass and JavaScript files via Bower, Composer, Meteor, or npm. Package managed installs don't include documentation, but do include our build system and readme.
npm install bootstrap@4.0.0-beta
gem install bootstrap -v 4.0.0.alpha6
bower install bootstrap#v4.0.0-beta
Read installation docs
Bootstrap CDN
When you only need to include Bootstrap's compiled CSS or JS, you can use the Bootstrap CDN.
CSS only
JS, Popper, and jQuery
Explore the docs
Official Themes
Take Bootstrap 4 to the next level with official premium themes—toolkits built on Bootstrap with new components and plugins, docs, and build tools.
Browse themes
GitHub Twitter Examples About
Designed and built with all the love in the world by @mdo and @fat. Maintained by the core team with the help of our contributors.
Currently v4.0.0-beta. Code licensed MIT, docs CC BY 3.0.

Figura 1.5: GetBootstrap.com

Voltando ao *navbar*, para encontrá-lo, basta acessar a documentação e pesquisar `Navbar` . Ao lado da busca, descendo um pouco, veremos todo o código do menu mais completo que o Bootstrap possui. Mas será que precisamos de tudo isso?

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupported(
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#">Disabled</a>
      </li>
    </ul>
    <form class="form-inline my-2 my-lg-0">
      <input class="form-control mr-sm-2" type="text" placeholder="Search" aria-label="Search">
      <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>
    </form>
  </div>
</nav>
```

Figura 1.6: Menu com muita coisa

Em nosso layout, não precisamos de tanta coisa. Para atingirmos nosso objetivo, bastaria a metade da esquerda. De estrutura HTML, o que vemos de semelhante entre a documentação e o nosso código? Os elementos `<nav>` e `<ul>` . Repare em suas classes:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  …
  <ul class="navbar-nav mr-auto">
```

Vamos experimentar colocar essas classes em nosso código HTML, da seguinte forma:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <ul class="navbar-nav mr-auto">
    <li><a href="#">Better</a></li>
```

```
    <li><a href="#">Descubra</a></li>
    <li><a href="#">Preços</a></li>
    <li><a href="#">Cadastre-se</a></li>
    <li><a href="#">Login</a></li>
  </ul>
</nav>
```

Abrindo no browser, com a janela maximizada, teremos:

Figura 1.7: Menu um pouco melhor

Um embaixo do outro

Se o seu projeto não ficou igual ao da imagem anterior, provavelmente ficou com os links um embaixo do outro. Maximize a janela do seu browser. É necessário que ela esteja com cerca de 1.000 pixels de largura.

Agora, os links estão grudados, mas uma pequena margem resolveria o problema. Assim, vamos usar tudo o que o Bootstrap tem de pronto para resolver isso. Na documentação, repare que cada link do menu possui a classe `nav-link` , então, façamos isso:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <ul class="navbar-nav mr-auto">
    <li><a href="#" class="nav-link">Better</a></li>
    <li><a href="#" class="nav-link">Descubra</a></li>
    <li><a href="#" class="nav-link">Preços</a></li>
    <li><a href="#" class="nav-link">Cadastre-se</a></li>
    <li><a href="#" class="nav-link">Login</a></li>
  </ul>
```

```
</nav>
```

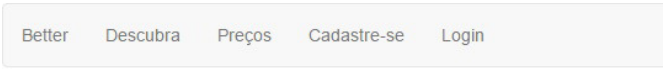


Figura 1.8: Menu bem melhor

Agora chegamos a um ponto interessante. Perceba como não foi preciso criar nenhuma regra CSS, simplesmente colocamos as classes informadas na documentação. Faz parte da rotina de um desenvolvedor que está criando algo com Bootstrap deixar o `getbootstrap.com` aberto, e é claro que, com o tempo, acabamos memorizando algumas delas.

Um outro detalhe dessa barra de navegação: reparou que o menu não está centralizado como a proposta do layout?

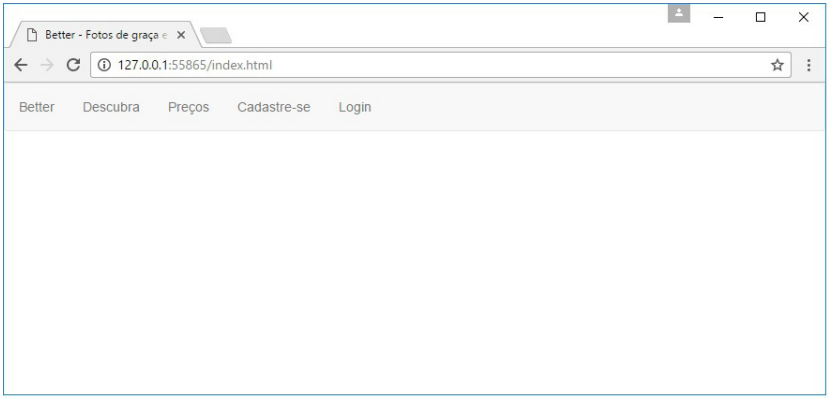


Figura 1.9: Menu encostado demais à esquerda

Como resolver isso? O Bootstrap possui uma classe genérica muito utilizada em diversas ocasiões que centraliza horizontalmente os elementos na tela, a classe `container` . Logo, só precisamos colocá-la envolvendo nosso menu:

```
<div class="container">
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <ul class="navbar-nav mr-auto">
      <li><a href="#" class="nav-link">Better</a></li>
      <li><a href="#" class="nav-link">Descubra</a></li>
      <li><a href="#" class="nav-link">Preços</a></li>
      <li><a href="#" class="nav-link">Cadastre-se</a></li>
      <li><a href="#" class="nav-link">Login</a></li>
    </ul>
  </nav>
</div>
```

Testando o código, chegaremos a este resultado:

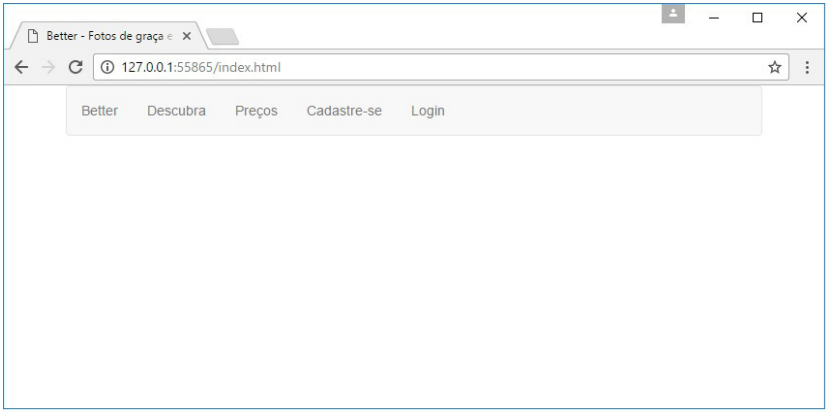


Figura 1.10: Cadê o resto das laterais?

Pegando a largura inteira

Caso você esteja fazendo outro projeto e precise de que a largura seja 100%, existe a classe `container-fluid` .

Repare que o menu foi centralizado, entretanto, ele está

cortando as laterais. Precisamos centralizar o `<nav>` todo ou só seu conteúdo? Dê uma checada no *wireframe* no começo do capítulo.

Somente o conteúdo deve ser centralizado, então, vamos mudar a `container` de lugar e colocá-la somente na `<ul>`, desta forma:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container">
    <ul class="navbar-nav mr-auto">
      <li><a href="#" class="nav-link">Better</a></li>
      <li><a href="#" class="nav-link">Descubra</a></li>
      <li><a href="#" class="nav-link">Preços</a></li>
      <li><a href="#" class="nav-link">Cadastre-se</a></li>
      <li><a href="#" class="nav-link">Login</a></li>
    </ul>
  </div>
</nav>
```

Chegando, agora sim, a um resultado muito próximo da proposta do site:

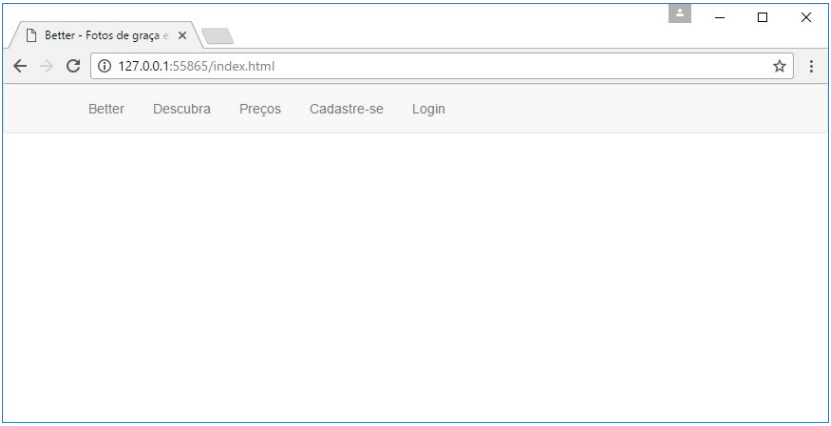


Figura 1.11: Menu certo e centralizado

1.1 E NO MOBILE?

Uma preocupação que tomou conta do mundo de desenvolvimento e design web foi o tal do *mobile*. Isso foi depois de os smartphones se popularizarem, lá em 2007. Logo, alguns devs pensaram ser interessante iniciar a concepção de um site tendo em mente uma tela de tamanho reduzido. E foi assim que surgiu o conceito de pensar primeiro no mobile, o *mobile first*.

É por conta desse conceito que o Bootstrap, a partir de sua versão 3, começou a focar primeiro no mobile. Com isso, daqui para a frente, vamos nos preocupar em como o projeto está ficando no celular também.

Diminua a tela do browser e repare como o menu ocupa muito espaço nela:

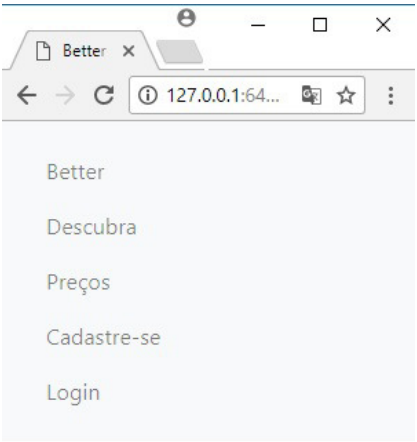


Figura 1.12: Menu todo aberto no mobile

Normalmente, quando estamos navegando em sites com o

celular, como o menu aparece? Vejamos:

Figura 1.13: Menus fechados

Vamos fazer com que o menu fique fechado (ou *colapsado*, em linguagem técnica). Caso o usuário precise utilizá-lo, ele clicará em um botão escrito `Menu`, para abrir a sua navegação. Primeiramente, vamos deixar o menu colapsado e nos preocupar com o botão em seguida.

Veja na documentação do Bootstrap, na parte do `navbar`, que ele utiliza duas classes chamadas `collapse` e `navbar-collapse`, deixando nosso código da seguinte forma:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container">
    <div class="collapse navbar-collapse">
      <ul class="navbar-nav mr-auto">
        <li><a href="#" class="nav-link">Better</a></li>
        <li><a href="#" class="nav-link">Descubra</a></li>
        <li><a href="#" class="nav-link">Preços</a></li>
        <li><a href="#" class="nav-link">Cadastre-se</a></li>
```

```
          <li><a href="#" class="nav-link">Login</a></li>
        </ul>
      </div> <!-- fim .collapse -->
    </div> <!-- fim .container -->
  </nav>
```

> **DICA**
>
> Agora é uma boa hora para começarmos a comentar nosso código. Use a convenção que preferir, mas colocarei comentários nas *tags* de fechamento de alguns elementos a fim de não nos perdermos durante o desenvolvimento do projeto.

Agora, o menu desaparece quando o browser está pequeno. Excelente, vamos criar o botão que o usuário usará para interagir com a navegação, e deixar o menu visível quando isso acontecer. Para se fazer um botão usando HTML, temos o elemento `<button>`. Vamos colocá-lo em nosso código, com o texto `Menu` em seu conteúdo:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container">
    <button>
      Menu
    </button>
    <div class="collapse navbar-collapse">
      <ul class="navbar-nav mr-auto">
        <li><a href="#" class="nav-link">Better</a></li>
        <li><a href="#" class="nav-link">Descubra</a></li>
        <li><a href="#" class="nav-link">Preços</a></li>
        <li><a href="#" class="nav-link">Cadastre-se</a></li>
        <li><a href="#" class="nav-link">Login</a></li>
      </ul>
    </div> <!-- fim .collapse -->
```

```
  </div> <!-- fim .container -->
</nav>
```

Para estilizá-lo, temos diversas classes que podem nos ajudar, como a `btn`, por exemplo, mas o posicionamento do botão ficaria assim:

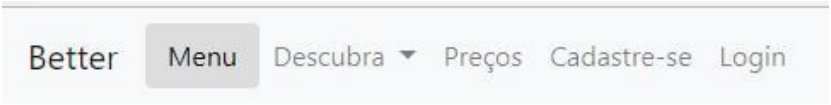


Figura 1.14: Botão errado do menu

Poderíamos colocar um `float: right` e fazer ajustes utilizando margens, entretanto, o Bootstrap possui uma classe para esse caso específico. Basta utilizarmos a classe `navbar-toggler`, que faz exatamente isso:

```
<button class="navbar-toggler">
  Menu
</button>
```

Maravilha, assim, o usuário já consegue utilizar o menu, certo? Tente clicar no botão e veja que, apesar da sua presença, o menu não está funcional... Ainda.

Colapsando muito cedo

Se você achou que o menu foi ocultado muito cedo, há uma solução. No `<nav>` principal, substitua a classe `navbar-expand-lg` pela `navbar-expand-sm`.

Veremos mais sobre essa relação de tamanhos do Bootstrap no capítulo *3. Entendendo as grids do Bootstrap*, que é focado no uso das *grids*.

Resta-nos agora a parte comportamental. Para a interação, sabemos que o JavaScript é uma boa pedida. O Bootstrap, além do arquivo de estilos, vem com um arquivo contendo plugins prontos, componentes que foram pensados para serem funcionais.

Então, precisaremos aprender JS? Não, basta chamarmos esse *script* e configurar os componentes devidamente. Vamos chamar o *script* do Bootstrap lá embaixo, antes de fechar a tag `body`:

```
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

SCRIPT NÃO É NO <HEAD> ?

Há alguns anos, invocávamos os arquivos JS dentro do elemento `<head>` , porém, como podemos ter certeza de que o elemento que queremos manipular já foi carregado pelo browser? Pensando nisso, a prática atualmente mais usada é puxá-los no final do `body` .

Agora o *script* do Bootstrap precisa saber quem ele deve colocar no comportamento de chave, de liga-desliga, de *toggle*. Para isso, devemos inserir no `<button>` dois atributos customizados que foram idealizados para fazer justamente essa ponte entre o HTML e o JS do Bootstrap, o `data-toggle` e o `data-target` .

E é no valor do segundo que passamos quem que será afetado, quem é o alvo do efeito (no caso, o elemento que tiver a classe `navbar-collapse`). Repare que precisamos colocar o seletor com o ponto no começo:

```
<button class="navbar-toggler" data-toggle="collapse" data-target
".navbar-collapse">
  Menu
</button>
```

Um detalhe técnico importante aqui é que, se essa classe `navbar-collapse` tiver seu nome alterado ou for removida em uma futura versão do Bootstrap, o botão quebrará. Normalmente, quando vamos fazer essa parte de interação em um elemento de um site, é mais preventivo atrelar essa interação a um ID, e não a uma classe. É por esse motivo que, na documentação da atual

versão do Bootstrap, o valor recomendado do `data-target` é um ID:

```
<button class="navbar-toggler" data-toggle="collapse" data-target
"#navbarSupportedContent">
  Menu
</button>
```

Porém, quem seria o alvo de fato? Em qual elemento queremos colocar o efeito de abre-fecha? Na `<div>`, em volta da `<ul>`. Inclusive, essa `<div>` possui classes para estilo, como a `collapse`:

```
<div class="container">
  <div class="collapse navbar-collapse" id="navbarSupportedConten
t">
    <ul class="navbar-nav mr-auto">
      ...
    </ul>
  </div>
</div>
```

Script carregado, botão e menus prontos, mas ao testarmos... Nada. O que houve? O arquivo JavaScript do Bootstrap foi desenvolvido em cima de outro *script*, o famoso **jQuery**. Se o Bootstrap depende do jQuery, precisamos chamá-lo no HTML para que o browser possa interpretar seu código corretamente, antes da chamada do JS do Bootstrap:

```
  <script src="js/jquery.js"></script>
  <script src="js/bootstrap.min.js"></script>
</body>
```

Teste o menu e veja que agora ele, enfim, funciona.

CADÊ O HAMBÚRGUER?

Hoje em dia, muitos menus de celular ficam com aquele ícone de hambúrguer, justamente para representar visualmente que ali consta um menu. Se você prefere usá-lo em vez do microtexto `Menu`, substitua-o por `<span class="navbar-toggler-icon"></span>`.

Com relação a qual é o melhor, resumidamente, não existe bala de prata. Recomendo a leitura do artigo da *Booking* sobre o teste A/B que fizeram a respeito do tema: https://booking.design/would-you-like-fries-with-that-4edf46849380.

1.2 TERMINANDO O MENU

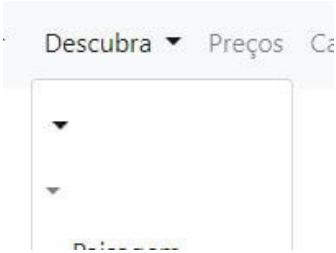
Progredimos bastante, mas ainda restam alguns itens a serem desenvolvidos a fim de ser o mais fiel possível ao layout:

- Submenu no item `Descubra`;
- Logo, ainda é preciso continuar visível no *mobile*.

Começando por este submenu, pergunto: o que é um submenu? Um menu dentro de um item de lista! Observe o menu a seguir da Alura Cursos Online de Tecnologia:

Figura 1.15: Submenu

Façamos a alteração no código colocando uma `<ul>` dentro da `<li>` de `Descubra`. Tomemos cuidado para **não colocar dentro do elemento `<a>`**. Caso você faça isso o menu ficará como a figura seguinte:




```
<li><a href="#" class="nav-link">Better</a></li>
<li>
  <a href="#" class="nav-link">
    Descubra
  </a>
  <ul>
    <!-- Aqui irão os itens do submenu -->
  </ul>
</li>
<li><a href="#" class="nav-link">Preços</a></li>
```

Esse submenu (a lista dentro do item de lista `Descubra`) possuirá seis itens, sendo as categorias de imagens da Better. Essas categorias foram passadas juntamente com o *briefing* do cliente. Diferente dos links do menu normais, vamos colocar a classe usada para identificar que estes links são de *dropdown*, com a classe `dropdown-item`:

```
<li><a href="#" class="nav-link">Better</a></li>
<li>
  <a href="#" class="nav-link">
    Descubra
  </a>
  <ul>
    <li><a href="#" class="dropdown-item">Paisagem</a></li>
    <li><a href="#" class="dropdown-item">Tecnologia</a></li>
    <li><a href="#" class="dropdown-item">Abstrato</a></li>
    <li><a href="#" class="dropdown-item">Animais</a></li>
    <li><a href="#" class="dropdown-item">Comida</a></li>
    <li><a href="#" class="dropdown-item">Pessoas</a></li>
  </ul>
</li>
<li><a href="#" class="nav-link">Preços</a></li>
```

Se você testar, verá que o submenu está sendo exibido por padrão, o que não faz sentido em nosso projeto. Além de especificar que cada link é um item de um *dropdown*, devemos colocar no código que a lista-pai desses elementos trata-se de um menu específico de *dropdown*, com a classe `dropdown-menu`.

```
<li><a href="#" class="nav-link">Better</a></li>
<li>
  <a href="#" class="nav-link">
    Descubra
  </a>
  <ul class="dropdown-menu">
    <li><a href="#" class="dropdown-item">Paisagem</a></li>
    <li><a href="#" class="dropdown-item">Tecnologia</a></li>
    <li><a href="#" class="dropdown-item">Abstrato</a></li>
    <li><a href="#" class="dropdown-item">Animais</a></li>
    <li><a href="#" class="dropdown-item">Comida</a></li>
```

```
    <li><a href="#" class="dropdown-item">Pessoas</a></li>
  </ul>
</li>
<li><a href="#" class="nav-link">Preços</a></li>
```

Não funciona, não é? Boa parte dessas interações do Bootstrap, como o menu hambúrguer, requer a importação do jQuery e do `Bootstrap.JS` para que elas funcionem e, além disso, a configuração da maneira correta. Nessa situação, vamos utilizar o já usado `data-toggle`, mas agora informando que ele será um *dropdown*:

```
<li><a href="#" class="nav-link">Better</a></li>
<li>
  <a href="#" class="nav-link" data-toggle="dropdown">
    Descubra
  </a>
  <ul class="dropdown-menu">
    <li><a href="#" class="dropdown-item">Paisagem</a></li>
    <li><a href="#" class="dropdown-item">Tecnologia</a></li>
    <li><a href="#" class="dropdown-item">Abstrato</a></li>
    <li><a href="#" class="dropdown-item">Animais</a></li>
    <li><a href="#" class="dropdown-item">Comida</a></li>
    <li><a href="#" class="dropdown-item">Pessoas</a></li>
  </ul>
</li>
<li><a href="#" class="nav-link">Preços</a></li>
```

Testando no browser, quando clicamos em `Descubra`:

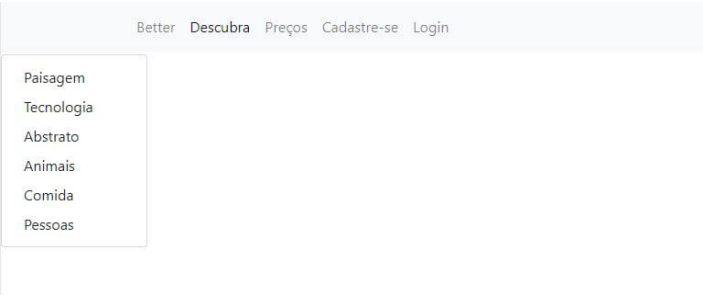


Figura 1.17: Dropdown ainda estranho

Opa! Os subitens aparecem, porém estão totalmente desconexos de seu elemento-pai. Para corrigir isso, devemos adicionar a classe que conecta o submenu, com seu respectivo item de lista, com a classe `dropdown`:

```
<li><a href="#" class="nav-link">Better</a></li>
<li class="dropdown">
  <a href="#" class="nav-link" data-toggle="dropdown">
    Descubra
  </a>
  <ul class="dropdown-menu">
    <li><a href="#" class="dropdown-item">Paisagem</a></li>
    <li><a href="#" class="dropdown-item">Tecnologia</a></li>
    <li><a href="#" class="dropdown-item">Abstrato</a></li>
    <li><a href="#" class="dropdown-item">Animais</a></li>
    <li><a href="#" class="dropdown-item">Comida</a></li>
    <li><a href="#" class="dropdown-item">Pessoas</a></li>
  </ul>
</li>
<li><a href="#" class="nav-link">Preços</a></li>
```

Então, como o usuário saberá que existe um menu dentro do item `Descubra`? É comum as interfaces de submenu possuírem alguma indicação visual de que ali há algo a mais. No nosso caso, podemos criar uma setinha, adicionando a classe `dropdown-toggle` ao link de `Descubra`:

```
<li>
  <a href="#" class="nav-link dropdown-toggle" data-toggle="dropd
own">
    Descubra
  </a>
  <ul class="dropdown-menu">
    <!-- Itens do submenu -->
  </ul>
</li>
```

Com isso feito, o submenu aparece quando clicamos no item `Descubra`. Sucesso!

Divisor de menus

A fim de separar os itens do submenu em categorias, podemos colocar um divisor de três em três itens. Isso pode ser feito facilmente ao colocarmos a classe `dropdown-divider` em uma `<li>` sem conteúdo. Ficaria desta forma:

```
<li><a href="#" class="nav-link">Abstrato</a></li>
<li class="dropdown-divider"></li>
<li><a href="#" class="nav-link">Comida</a></li>
```

Ótimo pela facilidade, mas um pouco triste pela semântica, por deixarmos um elemento vazio ali.

Repare que o logo da Better desaparece quando reduzimos o tamanho da janela do browser. Oras, se toda a `<div class=”collapse navbar-collapse>` é colapsada no mobile, podemos concluir que o logo está no local errado.

Para que ele não suma quando a tela estiver pequena, devemos isolá-lo acima dessa `div` , da forma vista a seguir. Inclusive, não faz mais sentido deixá-la em um item de lista, pois ela não está mais dentro de uma `<ul>` :

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a href="#" class="nav-link">Better</a>
  <button class="navbar-toggler" data-toggle="collapse" data-targ
et="#navbarSupportedContent">
    Menu
  </button>
  <div class="container">
    <div class="collapse navbar-collapse" id="navbarSupportedCont
ent">
      <ul class="navbar-nav mr-auto">
```

```
        <li class="dropdown dropdown-divider">
          <a href="#" class="nav-link dropdown-toggle" id="navbar
DropdownMenuLink" data-toggle="dropdown">
            Descubra
          </a>
          <ul class="dropdown-menu">
            ...
```

Com isso, o logo já não some, mas fica com a disposição visual estranha, parecendo apenas um link perdido:

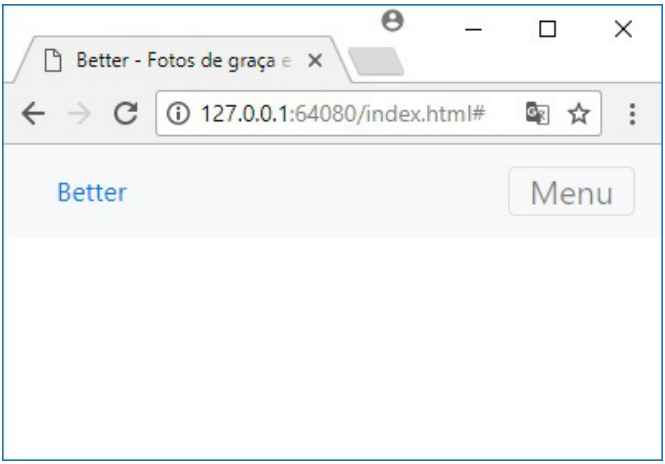


Figura 1.18: Logo na esquerda, mas em azul cor de link

Para consertarmos isso, vamos substituir a classe `nav-link` pela `navbar-brand` . Traduzindo, seria algo como "marca da barra de navegação":

```
<a href="#" class="navbar-brand">Better</a>
```

Confira que o resultado está como esperado:

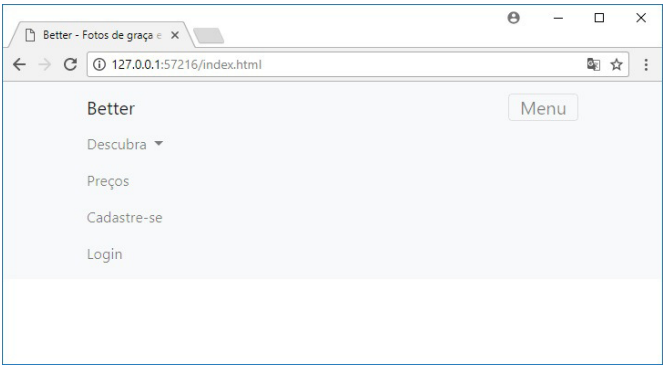


Figura 1.19: Menu conforme o esperado

Depois de tudo isso, o menu principal da Better, tanto mobile quanto desktop, estará esteticamente aceitável e funcionalmente interessante.

ALGO MAIS DARK

Achou o menu claro demais? Você pode alterar o esquema de cores facilmente apenas mudando as classes do `<nav>` . Dê uma olhada na documentação e divirta-se: https://getbootstrap.com/docs/4.0/components/navbar/#color-schemes.

1.3 RESUMO

Neste capítulo, vimos como o desenvolvimento web pode ser agilizado quando possuímos componentes prontos. Atualmente, a maior e mais conhecida biblioteca de componentes é o Bootstrap,

que basicamente é um arquivo CSS no qual basta chamarmos suas classes.

É importante frisar que não é necessário decorar as classes do Bootstrap. Além de serem muitas, em uma possível atualização da biblioteca, os nomes podem mudar totalmente. A documentação é a nossa melhor aliada ao trabalhar com Bootstrap, então, sempre fique com o *GetBootstrap.com* aberto em seu navegador.

Fazer um menu que se recolhe e se expande ao clique de um botão, sem ser necessário nenhum conhecimento de JavaScript ou de outras linguagens, pode ser uma tarefa fácil quando sabemos que o Bootstrap possui plugins JavaScripts prontos para uso. Apenas nos resta saber configurar tudo da forma correta e com os atributos correspondentes, todos muito bem explicados em sua documentação. Assim, até mesmo um submenu vira uma tarefa mais rápida e simples.

Dando continuidade ao projeto Better, a seguir, faremos a parte em que o usuário da plataforma mais vai interagir quando entrar na página principal, a busca.

Capítulo 2

DESTACANDO O QUE DEVE SER DESTACADO

Um conceito de que gosto bastante quando falo sobre usabilidade em cursos e palestras é o da dobra, ou *fold*. A ideia da dobra, quando estamos falando de sites, é uma linha imaginária em uma página que separa o que é carregado na tela do usuário **antes** de ele precisar fazer o *scroll* para baixo.

Figura 2.1: Exemplo de dobra

Pensando na UX da Better, algo interessante a ser feito é priorizar o que o usuário fará no site na parte acima da dobra, chamada também de *above the fold*. Este conceito é herança dos jornais impressos, que vinham dobrados, com as notícias mais importantes presentes acima de sua dobra para instigar o leitor a querer abri-lo.

Vamos aplicar esse conceito na Better!

USUÁRIO QUE NÃO USA O SCROLL DO MOUSE

Apesar do conceito da dobra fazer sentido, atualmente a ideia de que o usuário não desça a tela acabou tornando-se um mito. Você pode ler mais a respeito desse e outros mitos de UX no site: http://uxmyths.com.

2.1 RELEMBRANDO O WIREFRAME E COLOCANDO A MÃO NA MASSA

Vamos dar outra olhada no *wireframe*, já apresentado no começo do capítulo anterior, mas agora focando apenas em sua parte superior:

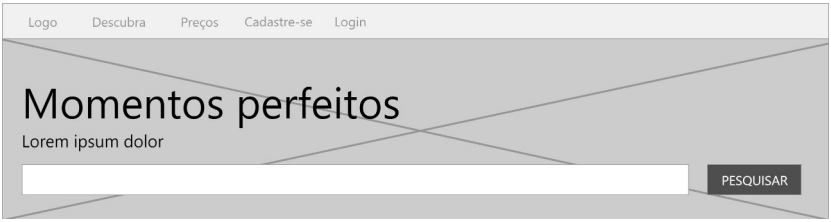


Figura 2.2: Wireframe parte de destaque

Já fizemos o menu, então, repare que temos alguns grandes elementos nesta seção de destaque:

- Título;
- Subtítulo;
- Campo de busca com botão.

Também temos um fundo para dar mais importância e tentar chamar mais atenção do usuário para essa parte.

Vamos começar a "codar" escrevendo o conteúdo e marcando com os devidos elementos HTML, **abaixo do menu (elemento `<nav>`)**, que fizemos anteriormente:

```
Momentos perfeitos
Lorem ipsum dolor
Buscar
```

Agora, marcaremos o conteúdo com HTML. O texto `Momentos perfeitos`, por exemplo, tem uma característica visual que remete a um parágrafo comum? Não. Pelo tamanho, parece um título; vamos então marcá-lo com um H1. Mas o texto abaixo dele remete a isso, logo, podemos usar um H2 ou um P:

```
<h1>Momentos perfeitos</h1>
<p>Lorem ipsum dolor</p>
```

Em seguida, façamos o campo de busca. Como estamos fazendo um código com HTML semântico, vamos colocá-lo juntamente com seu botão dentro de um `<form>`, da seguinte forma:

```
<h1>Momentos perfeitos</h1>
<p>Lorem ipsum dolor</p>
<form>
  <input type="search">
  <button>Pesquisar</button>
```

```
</form>
```

Por enquanto, o resultado é este:

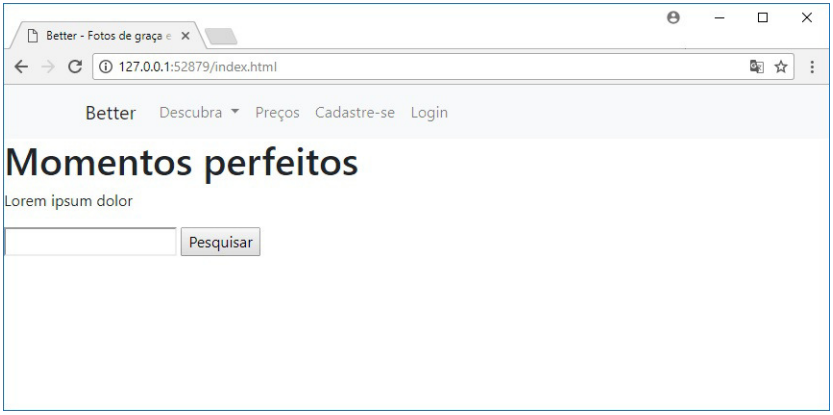


Figura 2.3: Destaque sem muita classe

Além de estar bem diferente da proposta do wireframe, podemos notar logo de cara que o conteúdo não está centralizado. É possível resolver isso com a classe `container` do Bootstrap, vista no capítulo anterior. Para isso, precisaremos criar um elemento em volta do nosso novo código.

Poderíamos colocar uma simples e genérica `<div>`, porém, o destaque faz parte do conteúdo principal do site. Logo, vamos usar o elemento do HTML5 criado especificamente para isso, o `<main>`:

```
<main class="container">
  <h1>Momentos perfeitos</h1>
  <p>Lorem ipsum dolor</p>
  <form>
    <input type="search">
    <button>Pesquisar</button>
  </form>
</main>
```

Como esperado, o conteúdo passa a ficar centralizado:

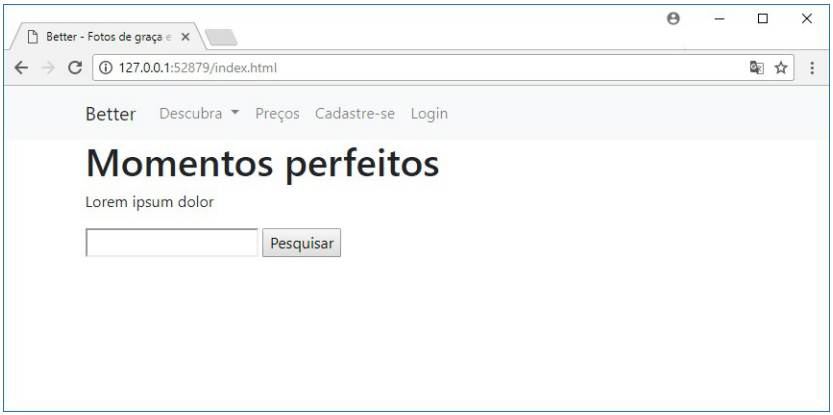


Figura 2.4: Destaque centralizado

Mas o conteúdo ainda parece menor que na proposta. Quão comum é deixarmos algo com um belo destaque em nossas páginas, e com um título em evidência e um bom espaço negativo em volta dele? Bastante. Tão comum que o Bootstrap possui uma classe até para isso, a `jumbotron`

2.2 JUMBO QUEM?

Um dos primeiros componentes que aprendi ao estudar Bootstrap foi, sem dúvida, o `jumbotron` . Na época, eu não sabia, mas esta palavra remetia àqueles televisores gigantes no meio das quadras de *baseball* e basquete. Ao pensarmos nisso, a ideia dela faz mais sentido ainda, isto é, dar um destaque imenso para algo importante.

Vamos brincar um pouco com ela. Primeiro, tente colocá-la em nosso elemento `<main>` :

```
<main class="container jumbotron">
  <h1>Momentos perfeitos</h1>
  <p>Lorem ipsum dolor</p>
  <form>
    <input type="search">
    <button>Pesquisar</button>
  </form>
</main>
```

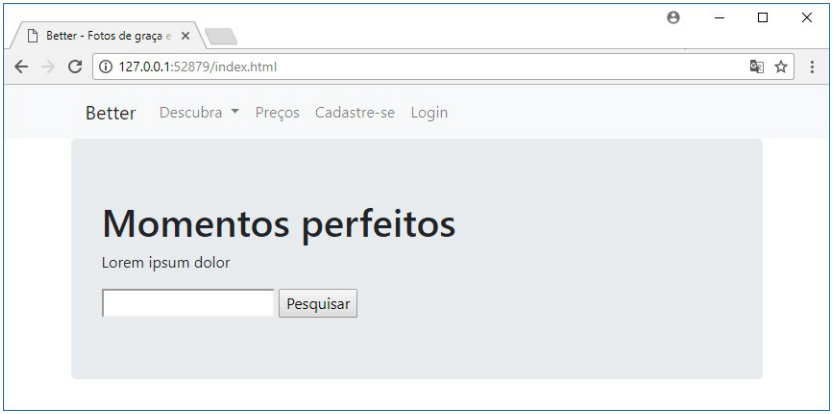


Figura 2.5: Jumbotron aplicada à main

Veja alguns pontos a serem observados com relação ao `jumbotron`:

1. Ela, de fato, aumenta o tamanho de todo o seu conteúdo;
2. Apareceu um cinza de fundo, que teremos de mudar posteriormente para uma imagem, segundo o *wireframe*.

Vamos resolvê-los individualmente. O fundo cinza vem por padrão em qualquer elemento que possua a classe `jumbotron`. Mas como normalmente colocamos uma imagem de fundo em nossos sites? Via CSS, com `background-image`!

Agora, o que fazer? Alterar a classe direto no Bootstrap ou criar

uma classe nossa, específica para isso? A opção mais rápida e prática é a primeira opção. Abrindo o arquivo `bootstrap.css`, vamos até a linha **4080**.

> **Flexibilidade**
>
> A maioria dos editores possui um atalho para irem a uma linha específica. No Sublime, por exemplo, use `Ctrl + G`, digite a linha que deseja ir e dê `Enter`.
>
> Ao conhecer bem sua IDE, você acaba ganhando velocidade na hora de desenvolver. Procure listas de atalhos na internet sobre o editor que estiver usando.

Agora, vamos modificar essa regra CSS do `jumbotron`:

```
.jumbotron {
  padding: 2rem 1rem;
  margin-bottom: 2rem;
  background-color: #e9ecef;
  border-radius: 0.3rem;
}
```

Basta adicionarmos a declaração ali no fim para fazer a imagem de fundo, como faríamos normalmente:

```
.jumbotron {
  padding: 2rem 1rem;
  margin-bottom: 2rem;
  background-color: #e9ecef;
  border-radius: 0.3rem;
  background-image: url(../img/vitrine.jpg);
}
```

Se o caminho da imagem estiver correto, estará tudo

funcionando. Logo, alterar o que precisamos direto no Bootstrap é uma prática que funciona e não machuca ninguém. Será?

Imagine um cenário no qual faremos diversas modificações no arquivo CSS original do Bootstrap, como alterar o tamanho dos títulos, cores padrões etc. Se sair uma nova versão do Bootstrap e atualizarmos esse arquivo, perderemos todas as nossas modificações. Por esse motivo, temos a primeira boa prática sobre Bootstrap: **não mexa no arquivo do Bootstrap**. Você até pode, mas será que vale o risco de dar um retrabalho para o *você-do-futuro*?

Então, como procedemos nessa situação? Manter o código 100% igual ao original engessaria muito o uso do framework. Uma característica bacana do CSS é poder **sobrescrever propriedades**, como no exemplo a seguir:

```
.jumbotron {
  background-color: #e9ecef;
}


.jumbotron {
  background-color: #BADA55
}
```

Mesma regra, mesmo seletor e mesma propriedade? A cor nova de *background* ficaria `#BADA55`, e não precisaríamos remover a `#EEE` original. Podemos aplicar a mesma ideia no nosso projeto, separando essas regras de sobrescrita em arquivos diferentes, justamente para não entrarmos no mesmo problema de nunca mais poder mexer no arquivo do Bootstrap.

Vamos criar um arquivo CSS novo com essas regras de sobrescritas, do nosso próprio tema, o `better-tema.css`, e

colocar na pasta `CSS`. Não esqueça de *linká-lo* no `<head>` de nossa `index.html`, junto ao arquivo do Bootstrap, **exatamente** desta maneira:

```
<link rel="stylesheet" href="css/better-tema.css">
<link rel="stylesheet" href="css/bootstrap.css">
```

> **ORDEM CERTA?**
>
> Repare na ordem em que colocamos os links. O que aconteceria se invertêssemos? Convido-o agora a ter a famosa **febre por testar** – sempre, com tudo.
>
> Tente inverter a ordem dos links ao término deste capítulo e reflita o que está acontecendo. Quem está sobrescrevendo quem? Retomaremos isso ainda aqui, mas peço que **realmente teste**.

Agora, no arquivo `better-tema.css`, basta colocarmos a regra CSS original do `jumbotron` juntamente ao novo *background*:

```
.jumbotron {
  padding: 2rem 1rem;
  margin-bottom: 2rem;
  background-color: #e9ecef;
  border-radius: 0.3rem;
  background-image: url(../img/vitrine.jpg);
}
```

Precisamos de todas essas declarações? Falar duas vezes que o `border-radius` precisa ter 0.3rem é um desperdício de bytes e de memória do browser, então, deixemos apenas aquilo que

precisamos (no nosso caso, o `background-image`):

```
.jumbotron {
  background-image: url(../img/vitrine.jpg);
}
```

Verificando no browser, podemos ver que o cinza do `jumbotron` sai de cena dando lugar para nossa linda imagem de vitrine.

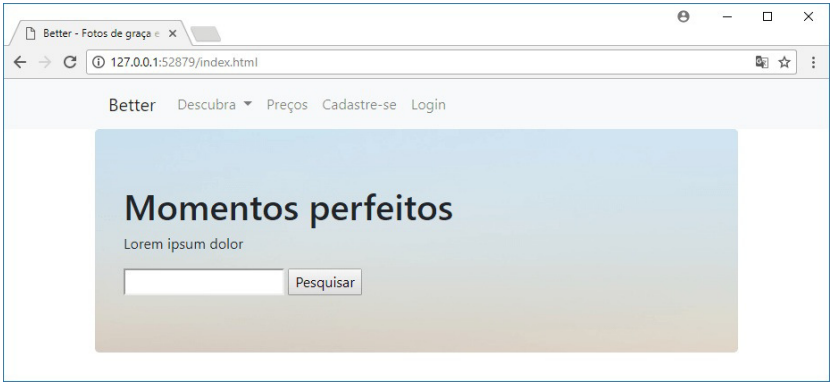


Figura 2.6: Jumbotron com degradê?

Está mais para um gradiente/degradê. Onde está o resto da imagem? Podemos conferir que o arquivo na pasta `imagens` está certo. Como corrigimos isso, então?

Repare que o tamanho da imagem parece estar muito grande para o espaço que a colocamos. A solução para isso seria diminuir o tamanho dela no Photoshop? Trabalhoso; seria mais fácil diminuirmos usando **apenas CSS**.

Na mesma regra que alteramos há pouco, vamos adicionar uma declaração para mudar o tamanho do `background` e posicioná-lo no centro do elemento:

```
.jumbotron {
  background-image: url(img/vitrine.jpg);
  background-size: cover;
  background-position: center;
}
```

Verifique o resultado:

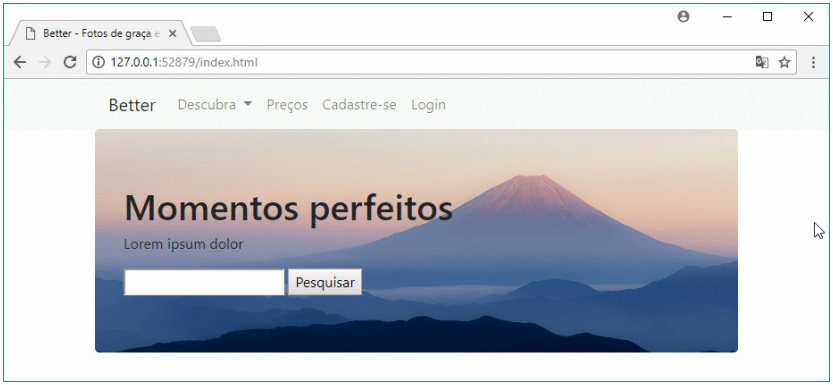


Figura 2.7: Jumbotron com imagem

Agora, sim, conseguimos ver a imagem. E caso seja necessário atualizar o Bootstrap para outra versão, nosso tema está totalmente a salvo em um arquivo CSS à parte!

2.3 TIRANDO AS MARGENS

Vamos resolver a questão das margens laterais do `jumbotron`. Seria bacana se a imagem esticasse, encostando nos limites da janela do browser.

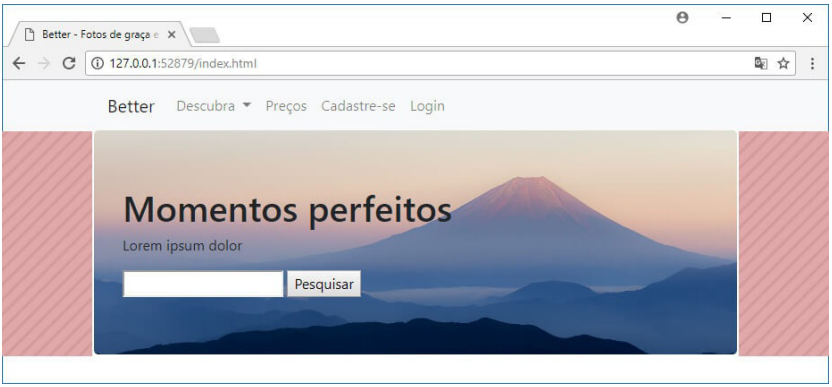


Figura 2.8: Margens ruins

Mas, de onde vem essas margens? Elas são uma margem colocada pelo `jumbotron`, ou um padding do `container`? Para descobrir, podemos ir ao arquivo do Bootstrap e procurar a classe que achamos que seria a responsável. Mas qual seria ela? Mais prático usar o inspetor de código do seu browser.

No meu caso, usarei o Devtools do Chrome, também chamado de *inspect element*, ou mesmo `F12`. Com a página aberta, basta clicar com o botão direito no elemento alvo – no nosso caso, na área de destaque, logo abaixo do menu –, e clicar na opção `Inspecionar` (ou *Inspect*):

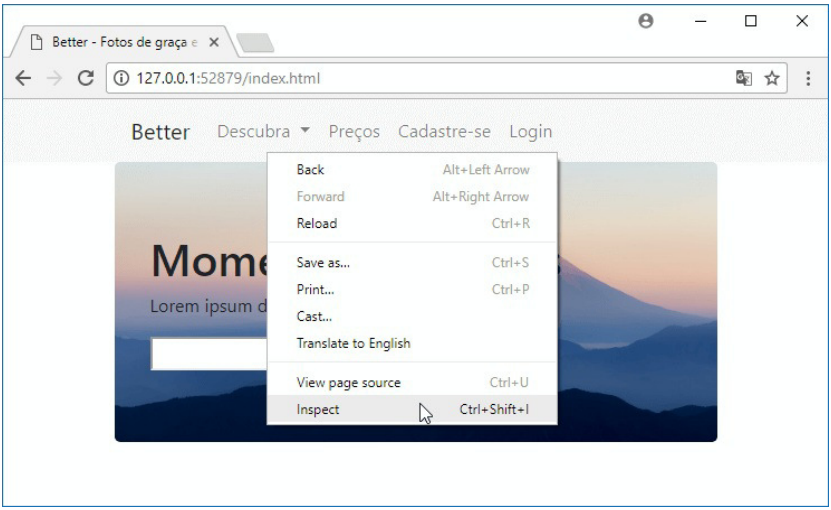


Figura 2.9: Inspecionar elemento

Em um dos cantos do Devtools, aparecerá o *box model* do elemento selecionado. Se não estiver vendo-o, clique na aba `Computed`.

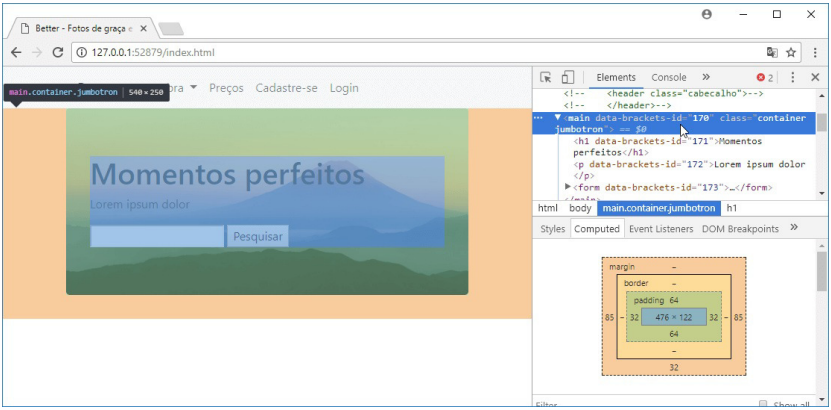


Figura 2.10: Caçando a margem

O gráfico à direita mostra os valores de conteúdo, padding, borda e margem aplicados a um determinado elemento – no caso, o selecionado é o `<main>` . Além disso, conforme você passa o mouse no código HTML à direita, ele vai colorindo o elemento correspondente na visualização do browser. Bacana, não?

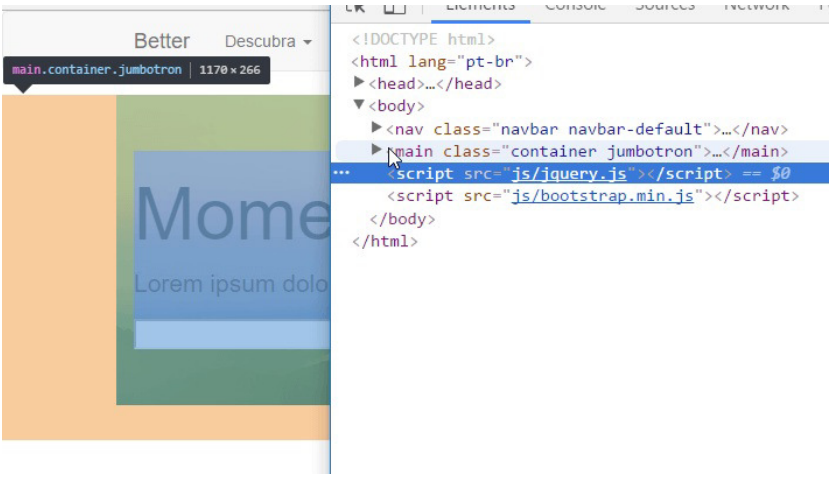


Figura 2.11: Devtools com foco no elemento main

Ainda no Devtools, com o `<main>` selecionado, vamos descer um pouco procurando de qual classe está vindo essa margem. Clicando na `margin-left` , veja que ela está sendo aplicada pela classe `container` :

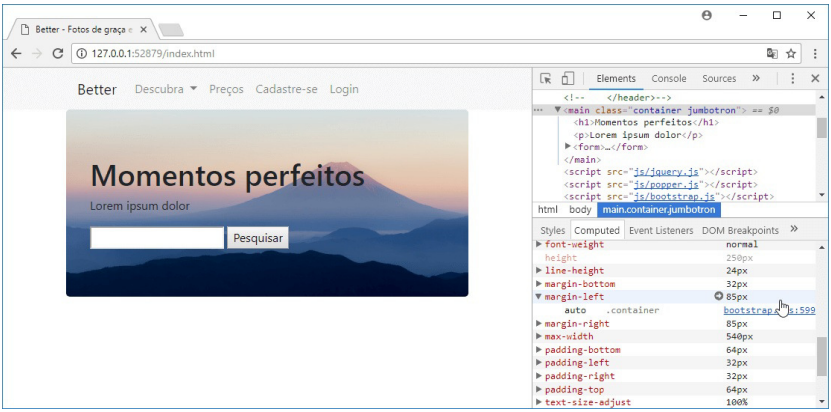


Figura 2.12: Devtools mostrando que margem vem da container

Observe que as margens laterais não eram culpa do `jumbotron`, mas sim da `container`. Você pode brincar com os valores dessas margens ali no gráfico, se desejar. Só tenha cuidado com o fato de que, por padrão, o Devtools perderá as alterações quando você atualizar a página.

DEVTOOLS

O Devtools, um dos maiores ajudantes do desenvolvedor web, precisa ser explorado por você, leitor, caso não o conheça. Recomendo que saia clicando e mudando valores; brinque à vontade, pois você não quebrará nada.

Para que as alterações sejam salvas, dê uma olhada neste post: https://www.turbosite.com.br/blog/15-dicas-para-desenvolver-utilizando-o-chrome-devtools/.

Outra forma de mudar os valores ali é usar o *inspect element* do navegador no elemento `<main>`. Basta marcar e desmarcar algumas propriedades, e analisar o resultado no próprio browser.

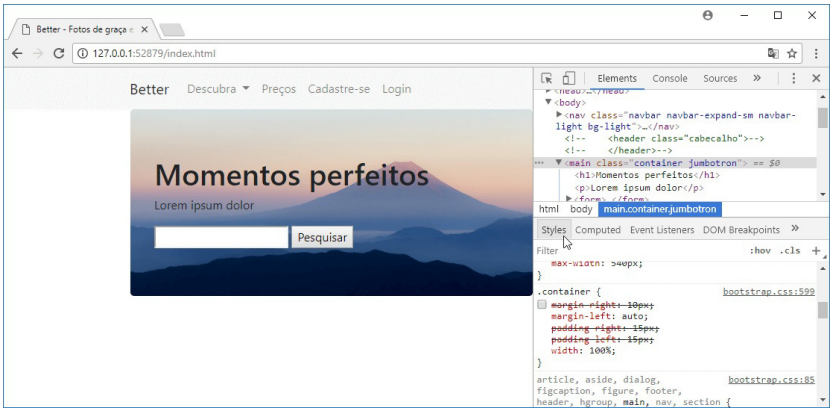


Figura 2.13: Devtools no main mostrando seu CSS

Se tirarmos as margens da `.container`, nosso menu seria afetado. Poderíamos criar uma classe específica sem as margens, algo como `container-do-destaque`, mas ela seria uma sombra da `.container` original. O que fazer?

Façamos um teste. No `index.html`, tire do `<main>` a classe que está nos complicando, a `.container`, deixando apenas a `jumbotron`.

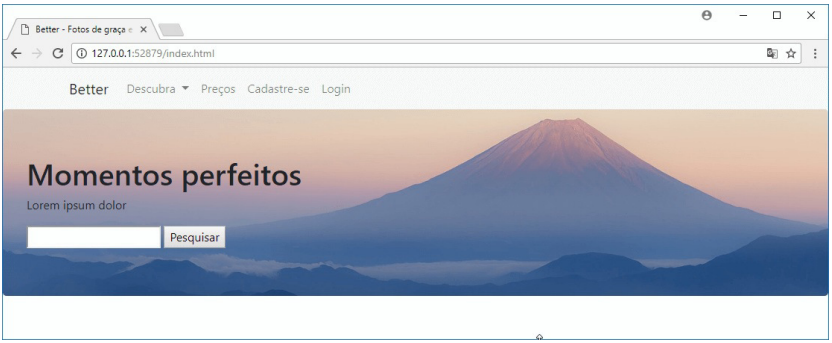


Figura 2.14: Sem container

Perceba que precisamos centralizar o destaque, mas o problema é que seria mais cômodo usar a `container`. Porém, será que é necessário centralizar o destaque inteiro ou **apenas o conteúdo** do destaque? Apenas o conteúdo!

Vamos mexer no nosso HTML e fazer com que o `jumbotron` agora seja o elemento-pai do elemento com o `container`:

```
<main class="jumbotron">
  <div class="container">
    <h1>Momentos perfeitos</h1>
    <p>Lorem ipsum dolor</p>
    <form>
      <input type="search">
      <button>Pesquisar</button>
    </form>
  </div>
</main>
```

Ao conferir o resultado no browser, teremos:

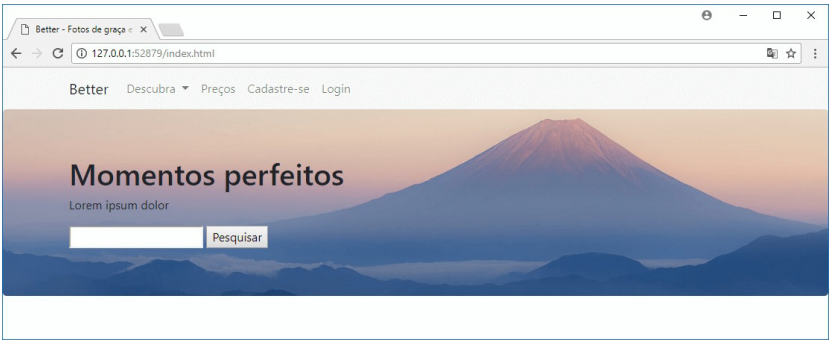


Figura 2.15: Destaque com container

Finalmente conseguimos! É importante notarmos que alterar o Bootstrap é relativamente fácil, mas, às vezes, o problema não está nele, e sim em nosso próprio código; nesse último caso, em nosso HTML.

2.4 ESTILIZANDO UM CAMPO DE FORMULÁRIO

Estilizar formulários com Bootstrap é extremamente prático. Com o uso das classes de campos que estão no framework, podemos fazer um extenso formulário com apenas algumas linhas de código. Foquemos agora em deixar o campo de busca com uma cara um pouco melhor. Por enquanto, ele está pequeno, sem graça e sem formatação nenhuma.

Veja o código responsável pela busca na `index.html`:

```
<form>
  <input type="search">
  <button>Pesquisar</button>
</form>
```

O primeiro passo é saber qual elemento precisamos estilizar. O campo de busca, certo? O `<input>` precisa ficar um pouco maior e com um design mais agradável. Vamos utilizar uma classe do Bootstrap chamada `form-control` para isso:

```
<form>
  <input type="search" class="form-control">
  <button>Pesquisar</button>
</form>
```

Com isso, repare que o campo começa a ficar mais interessante, com uma largura batendo nos limites da área do destaque, uma ligeira borda arredondada e uma altura maior. Experimente colocar o atributo `placeholder` para dar uma dica ao nosso usuário sobre o que ele deve fazer ali no campo:

```
<form>
  <input type="search" class="form-control" placeholder="Encontre
 o que você procura...">
  <button>Pesquisar</button>
</form>
```

> **O QUE MAIS VEM NA FORM-CONTROL?**
>
> Lembre-se de que a qualquer momento você pode (e deve) fazer uso do Devtools caso queira saber quais declarações são usadas em determinadas classes que estamos vendo no livro. Isso é uma prática excelente de estudo, principalmente se você já desenvolve ou tem forte interesse em front-end.

Além de campos de texto, qual é outro elemento que sempre encontramos em formulários? Botões. Não só em formulários, mas em grande parte das interfaces visuais o uso de botões é algo

bastante usado há um bom tempo, principalmente por fazer analogia ao mundo real, uma das heurísticas de usabilidade de Nielsen, referência mundial em usabilidade e UX.

Para estilizarmos nosso cinza e humilde botão, basta usarmos a classe `btn` do Bootstrap:

```
<form>
  <input type="search" class="form-control" placeholder="Encontre
 o que você procura...">
  <button class="btn">Pesquisar</button>
</form>
```

Com isso, ele já fica mais encorpado, mas continua cinza. Verifique no browser. Isso se deve à maneira como os componentes do Bootstrap são organizados. Existem classes para **estrutura** e classes para **estilização**, e, no caso, a `btn` é apenas para a estrutura. A estilização fica sendo responsabilidade de outras classes.

Mas qual é o motivo de se fazer tudo separado, e não apenas classes como `btn-azul` ou `btn-vermelho`? Temos dois problemas: o primeiro (e mais óbvio) é a nomenclatura de uma classe assim. Imagine uma chamada `coluna-esquerda` ou `texto-azul`. O que aconteceria se fosse necessário alterar a primeira para a `direita` e a segunda para outra cor? Preferencialmente, o layout não deve mandar em como nomeamos as nossas classes, e sim a estrutura. Dizemos que classes assim são sem semântica.

O Bootstrap possui cores padrões como azul, vermelho, verde etc. Mas, pensando no parágrafo anterior, o que aconteceria se mudássemos o *background* de um elemento com a classe `btn-azul` para vermelho? Perderíamos a semântica da classe, nada

bateria com nada e, eventualmente, o desenvolvedor que pegasse esse código para alguma manutenção provavelmente nos xingaria.

Sim, temos cores padrões no Bootstrap, mas, pensando nisso, ele não chama o azul de azul, por exemplo, e sim de cor primária! O `btn-primary` para ser mais exato. Façamos o teste em nosso código:

```
<form>
  <input type="search" class="form-control" placeholder="Encontre
 o que você procura...">
  <button class="btn-primary">Pesquisar</button>
</form>
```

Ficou azul, que é a cor primária no Bootstrap, porém, note que ele perdeu a formatação visual de estrutura. Como o Bootstrap organiza seus componentes mesmo? Estrutura e layout! Então, devemos deixar as duas classes, mesmo que à primeira vista seja redundante, já que é como a arquitetura do framework foi pensada:

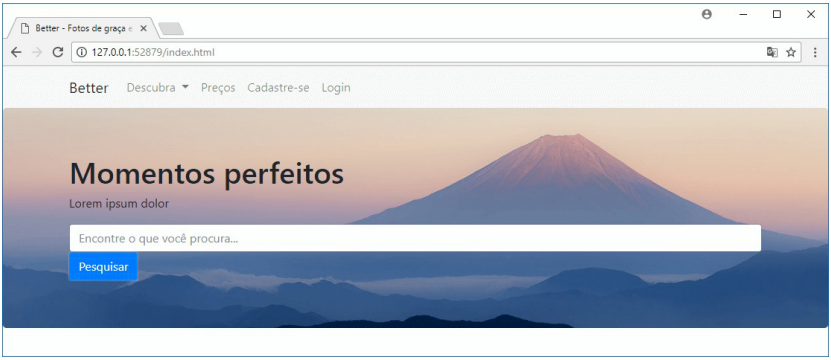
```
<form>
  <input type="search" class="form-control" placeholder="Encontre
 o que você procura...">
  <button class="btn btn-primary">Pesquisar</button>
</form>
```

Agora o botão ficou bacana. Repare também que ele possui a mesma borda arredondada do campo. Como você faria para deixar sem essa borda? Faça testes no `better-tema.css` .

Outras cores

A cor primária do Bootstrap é este tom de azul que vimos no browser há pouco. Procure na documentação outras cores/classes usadas no framework. Como a cor de perigo (vermelho) e de sucesso (verde). Brinque com elas, coloque outras cores.

Por enquanto, nossa área de destaque ficou desta maneira:



2.5 CAMPO E BOTÃO LADO A LADO

Um dos primeiros conceitos mais importantes que aprendi sobre front-end foi que cada elemento, quando interpretado pelo browser, era como se fosse uma caixinha. Juntamente com esse conceito, aprendi sobre os valores `inline` e `block` da propriedade *display*. Respectivamente, elementos *inline* ficam na mesma linha e não é possível colocar dimensões neles. Já os elementos de bloco ocupam a linha inteira na qual estão sendo

renderizados, sendo possível colocar a largura e a altura.

E se precisarmos de um elemento com dimensões inseridas por nós e, ao mesmo tempo, que fique um do lado do outro? É aí que entrará o outro valor da *display*, o `inline-block`.

Na última imagem, repare que o campo e o botão estão ocupando a linha toda, individualmente. O que precisamos fazer então? Colocá-los como `inline-block`, mas com Bootstrap. Vamos olhar o código novamente:

```
<form>
  <input type="search" class="form-control" placeholder="Encontre
 o que você procura...">
  <button class="btn btn-primary">Pesquisar</button>
</form>
```

Qual elemento genérico do HTML nos permite deixar elementos *inline*? O `<span>`, entretanto, ele sozinho de nada adiantaria. Precisamos usar a classe do Bootstrap `input-group-btn`, representando que o botão é a parte de um todo:

```
<form>
  <input type="search" class="form-control" placeholder="Encontre
 o que você procura...">
  <span class="input-group-btn">
    <button class="btn btn-primary">Pesquisar</button>
  </span>
</form>
```

Para sempre Bootstrap

O leitor já deve ter reparado que a maioria das classes que usamos até aqui é do Bootstrap. A partir de agora, será informado de onde veio a classe somente se ela não for do framework.

Se você olhar o resultado, provavelmente vai começar a ficar desesperado; não fique. Falamos que o botão faz parte de um todo, mas que todo? Precisamos dizer que o *input* e o botão precisam ficar dentro da mesma linha. Em nosso código HTML, como isso está sendo representado? Com o elemento `form` ! Basta colocarmos a classe `input-group` nele:

```
<form class="input-group">
  <input type="search" class="form-control" placeholder="Encontre
 o que você procura...">
  <span class="input-group-btn">
    <button class="btn btn-primary">Pesquisar</button>
  </span>
</form>
```

Pronto, agora nosso campo de busca e seu botão estão na mesma linha.

Outros usos da input-group

Este não é o único caso que esta classe pode ser usada. Procure na documentação por mais alguns exemplos e imagine como ela poderia ser aplicada a alguma interface que você está desenvolvendo atualmente. Acesse: http://getbootstrap.com/components/#input-groups.

Nossa área de destaque já está fiel à proposta, mas podemos melhorar um pouco nossa *user interface* (UI). Todo o texto que está dentro do *jumbotron* está preto; e se trocássemos a cor para branco, ele não teria um destaque maior? Façamos isso.

Será que o Bootstrap possui a classe `text-white`, e será que há uma classe para cada cor existente? O Bootstrap precisaria ter alguns gigabytes só para ter esse tipo de informação. Podemos verificar na documentação na parte de classes de ajuda, ou *helper classes* (http://getbootstrap.com/css/#helper-classes), que podemos mudar a cor dos textos baseando-se no que aquele texto representa. Esta é a mesma ideia dos botões com as classes `btn-primary` ou `btn-success`, que estão atreladas às cores padrões usadas pelo framework. Então, nada de cor branca.

Precisaremos fazer na mão, ou seja, sobrescrever a cor de texto do `jumbotron`. Já colocamos uma imagem de fundo em nosso `better-tema.css`, agora basta adicionarmos a declaração para deixarmos todo o texto inserido ali branco:

```
.jumbotron {
  background-image: url(/imagens/vitrine.jpg);
```

```
  background-size: cover;
  background-position: center;
  color: #fff;
}
```

> **Mas existe classe de texto branco sim!**
>
> Para fins didáticos, omiti essa informação. Porém, o Bootstrap 4 possui classes para **algumas** cores, então a teoria dos gigabytes por conta de classes específicas para cada cor se mantém.

Se a cor não estiver pegando, lembre-se de que é importante chamar o CSS do nosso tema **após** chamarmos o CSS do Bootstrap desta maneira, lá no `<head>`:

```
<link rel="stylesheet" href="/css/bootstrap.css">
<link rel="stylesheet" href="/css/better-tema.css">
```

> **Especificidade CSS**
>
> Recomendo ao leitor buscar conhecimento sobre o tema. Um bom texto a ser lido é o da apostila de HTML, CSS e JavaScript da Caelum: https://www.caelum.com.br/apostila-html-css-javascript/css-avancado/#5-17-para-saber-mais-especificidade-de-seletores-css.

Feito isso, nosso destaque está pronto. Fique à vontade para mudar o texto do parágrafo para algo mais convidativo, como

"Mais de 17 yottabytes de fotos".

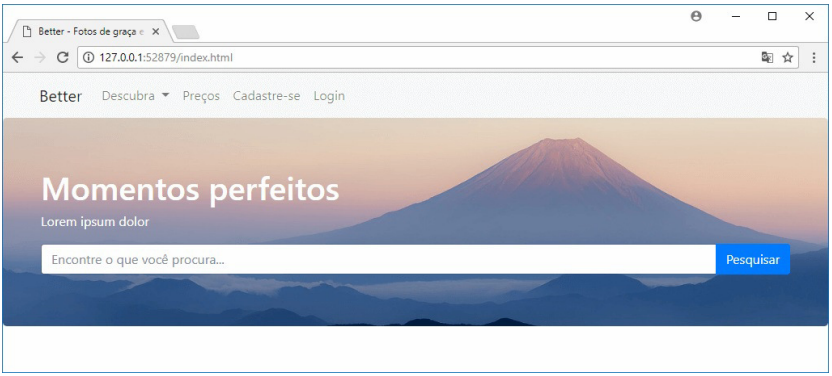


Figura 2.17: Destaque ok

2.6 RESUMO

Nosso projeto está pegando forma cada vez mais. Neste capítulo, vimos mais uma amostra do poder e praticidade que o Bootstrap nos proporciona, e usamos um famoso componente do framework, o Jumbotron, uma mão na roda para fazer áreas que precisam de um grande destaque em sites de forma rápida.

Mas sempre haverá a necessidade de customizar o Bootstrap, então, vimos por que editar direto o arquivo do Bootstrap pode não ser a melhor opção, e uma prática interessante de criar o próprio tema em um arquivo CSS a parte – para melhor controle e ganho de flexibilidade, com relação a possíveis atualizações do Bootstrap.

Um fiel escudeiro de qualquer desenvolvedor, sem dúvida, é o *developer tools* (ou Devtools). E quando estamos usando o Bootstrap, não é diferente. Fuçamos nosso código com ele para

pegarmos e entendermos o que o Bootstrap estava fazendo por de baixo dos panos. Além disso, vimos que a ordem das chamadas de nossos CSSs importa para o browser saber qual regra deve permanecer.

Por fim, fizemos nosso primeiro campo de formulário, estilizado totalmente com os padrões do Bootstrap. Entendemos como funciona a relação de classes de estrutura e classes de estilo com as `btn` e `btn-primary` de exemplo.

No próximo capítulo, vamos ver e mexer bastante com uma das *features* mais interessantes que o framework possui, as **grids**.

Capítulo 3

ENTENDENDO AS GRIDS DO BOOTSTRAP

Sabe quando você olha para um site e os elementos parecem não estar combinando com nada? Às vezes, você pode não saber exatamente o que há de errado com o layout, mas seu *eu-interior* grita: "Isso não está ornando não!". Muitas vezes, em situações como essa, o problema é a falta de harmonia no layout, falta uma padronização visual entre conteúdo, espaçamentos etc.

Além disso, o posicionamento de elementos em um projeto front-end é uma das dificuldades mais comuns, principalmente quando lidamos com layouts com várias colunas e que podem não respeitar muito uma hierarquia bem estruturada.

Isso pode ser facilitado e corrigido com um conceito que foi criado inicialmente para ser um aliado na área de design, mas que o mundo de front-end absorveu de forma eficaz, o fantástico conceito de *grids*.

A ideia é simples: dividir a tela em diversas colunas e ir encaixando os elementos dentro desse *grid*. E para nossa sorte, grande parte dos *frameworks* CSS possuem sistemas de *grids*, dos mais simples aos mais complexos. E com o Bootstrap não seria

diferente.

3.1 ANALISANDO O WIREFRAME DA GALERIA

A vontade de "codar" é tanta, que muitas vezes queremos já queimar largada e já partir para a codificação. É um sentimento justo, mas muitas vezes perigoso. Algo que pode evitar manutenções desnecessárias é analisar o que precisa ser feito, antes de fazê-lo. Façamos isso com a terceira grande seção da home da Better, a seção de galeria:

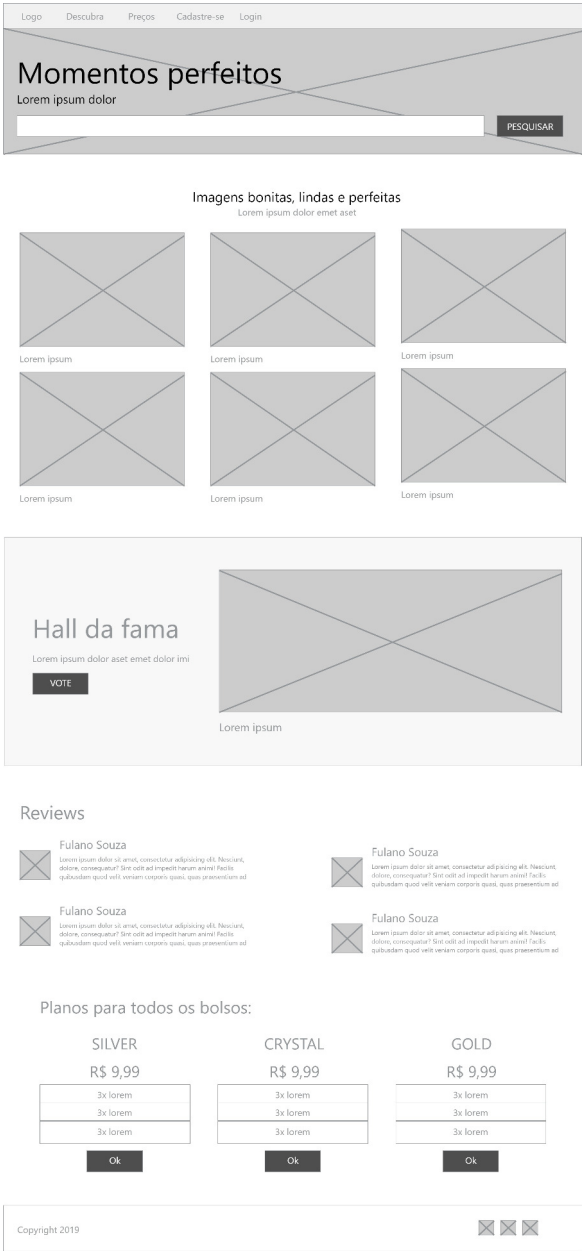


Figura 3.1: Wireframe da galeria

Voltando nosso modelo mental a partir de agora para pensar em colunas, repare que essa parte possui basicamente três grandes colunas de largura igual:

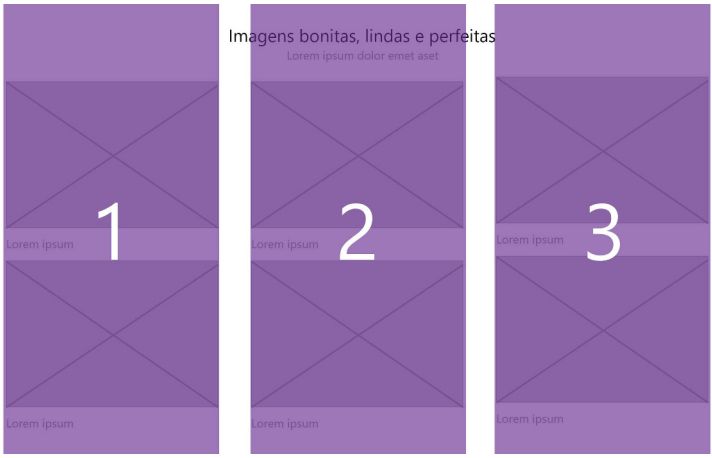


Figura 3.2: Colunas

3.2 COMEÇANDO A MARCAÇÃO

Vamos começar a codificar essa seção apenas marcando o conteúdo inicialmente, como fizemos nos capítulos anteriores. Inicialmente, coloquemos o conteúdo para depois ver como vamos marcá-lo. Na `index.html`, **dentro** do elemento `main` que fizemos antes:

```
  Imagens bonitas, lindas e perfeitas

  Lorem ipsum dolor emet aset
```

Como essa parte do site será uma área dedicada a fotos de galeria, podemos envolver esse conteúdo em uma `section`:

```
<section>
  Imagens bonitas, lindas e perfeitas
```

```
  Lorem ipsum dolor emet aset
</section>
```

O que essa frase "Imagens bonitas (...)" representa nessa seção? Seu título! Marquemos com um `h2`. No outro texto, podemos marcar apenas com um parágrafo ou mesmo um `h3`:

```
<section>
  <h2>
    Imagens bonitas, lindas e perfeitas
  </h2>
  <p>
    Lorem ipsum dolor emet aset
  </p>
</section>
```

Além desses textos, coloquemos também as seis imagens, e suas respectivas legendas, conforme o *wireframe*. Pense no usuário, o que ele fará com as imagens? Provavelmente clicará para abrir a página da imagem específica. Vamos colocar então alguns `<a>` envolvendo cada imagem também. As imagens que foram enviadas pelo cliente se encontram na pasta `imagens` dentro da pasta do projeto.

```
<section>
  <h2>
    Imagens bonitas, lindas e perfeitas
  </h2>
  <p>
    Lorem ipsum dolor emet aset
  </p>

  <a href="#">
    <figure>
      <img src="img/girassois.jpg" alt="Girassóis no campo">
      <figcaption>
        Girassóis
      </figcaption>
    </figure>
```

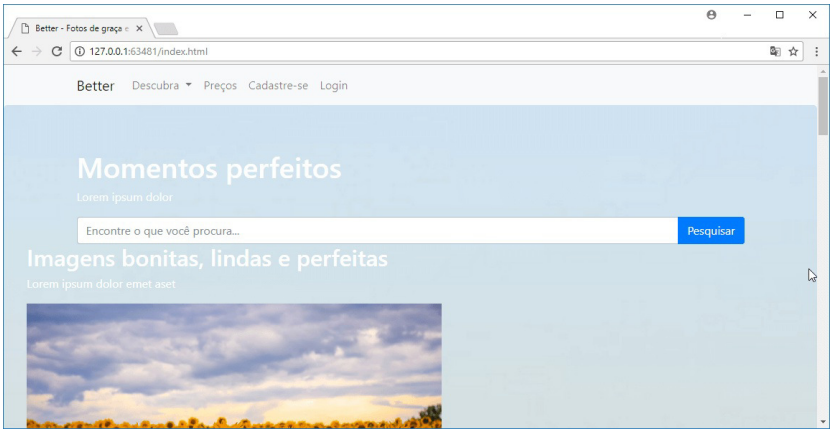
```
  </a>

</section>
```

Atenção nesse `<a>`, pois precisamos de seis dele no total. Basta copiar e colar seis vezes o `<a>` e apenas mudar o `src` e o `alt`, como também o conteúdo nas `figcaption`:

```
<a href="#">
  <figure>
    <img src="img/girassois.jpg" alt="Girassóis no campo">
    <figcaption>
      Girassóis
    </figcaption>
  </figure>
</a>

<a href="#">
    <figure>
    <img src="img/garota-silhueta.jpg" alt="Silhueta garota">
    <figcaption>
      Silhueta garota
    </figcaption>
  </figure>
</a>

<a href="#">
  <figure>
    <img src="img/mesa-designer.jpg" alt="Típica mesa de um desig
ner">
    <figcaption>
      Mesa de designer
    </figcaption>
  </figure>
</a>

<a href="#">
  <figure>
    <img src="img/pipoca.jpg" alt="Pipoca">
    <figcaption>
      Pipoca
    </figcaption>
  </figure>
```

```
</a>

<a href="#">
  <figure>
    <img src="img/garota-feliz.jpg" alt="Mulher feliz">
    <figcaption>
      Mulher feliz
    </figcaption>
  </figure>
</a>

<a href="#">
  <figure>
    <img src="img/sunset.jpg" alt="Pôr do sol">
    <figcaption>
      Pôr do sol
    </figcaption>
  </figure>
</a>
```

Feito isso, vamos ver o resultado no browser:



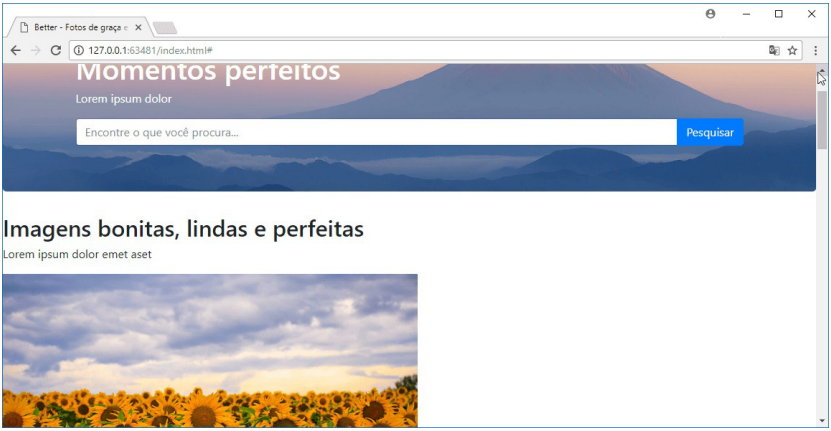
É como se as imagens estivessem dentro do `jumbotron`. Mas elas não deveriam ficar em uma seção abaixo dele?

Podemos resolver isso simplesmente colocando o `jumbotron`

em volta da parte de destaque apenas, e não do `main` inteiro. E para deixar o código mais semântico, é possível usar uma `<section>` para isso:

```
<main>
  <section class="jumbotron">
    <div class="container">
      <h1>Momentos perfeitos</h1>
      <p>Mais de um 17 yottabytes de fotos</p>

      <form class="input-group">
        <input type="search" class="form-control" placeholder="En
contre o que você procura...">
        <span class="input-group-btn">
          <button class="btn btn-primary">Pesquisar</button>
        </span>
      </form>
    </div>
  </section> <!-- fim .jumbotron -->
```

Teste no browser e veja que, agora, as imagens estão isoladas na parte de baixo, e o `jumbotron` está certo.



3.3 CENTRALIZANDO TUDO

Como fizemos para centralizar o conteúdo do `jumbotron` no capítulo anterior? Usamos a classe `container`. Façamos o mesmo nessa `<section>` recém-criada, relativa à nossa futura galeria de imagens:

```
<section class="container">
  <h2>
    Imagens bonitas, lindas e perfeitas
  </h2>
  <p>
    Lorem ipsum dolor emet aset
  </p>

  <!-- Aqui ainda é mantido os <a> das imagens -->
</section>
```

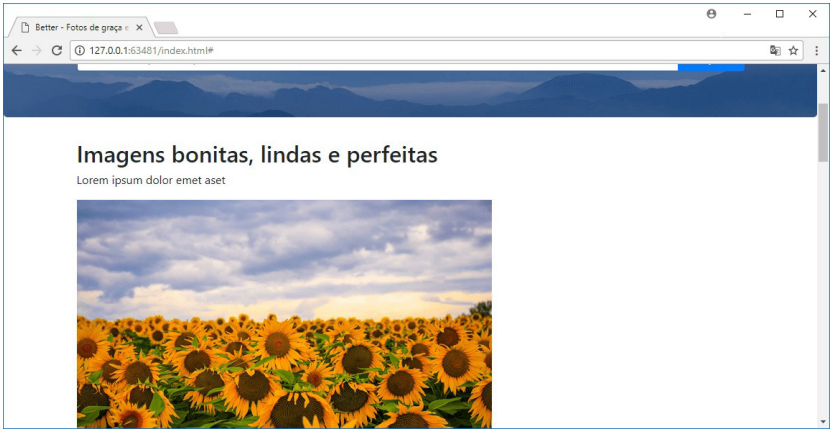


Figura 3.5: Container na section

Com o conteúdo centralizado, lembre-se de que, segundo o *wireframe*, os textos iniciais dessa seção também precisam estar centralizados. Podemos resolver isso usando uma das classes relacionadas ao alinhamento de texto do Bootstrap, a `text-center`:

```
<section class="container">
  <h2 class="text-center">
    Imagens bonitas, lindas e perfeitas
  </h2>
  <p class="text-center">
    Lorem ipsum dolor emet aset
  </p>

  <!-- Aqui ainda tem os <a> das imagens -->
</section>
```

Então, quando precisarmos centralizar textos, podemos usar essa classe que nada mais é que a declaração `text-align: center` por baixo dos panos. Como você já deve ter imaginado e testado, se quiséssemos alinhar em outras direções, também teremos outras classes, como a `text-justify` ou a `text-right`.

É importante notar também que, quando precisamos centralizar grandes blocos de conteúdo, usamos a `container`. Agora, para centralizar o conteúdo em si, como textos e imagens, usamos a `text-center`.

3.4 MELHORANDO A GALERIA

Tente redimensionar a tela de seu navegador diminuindo-a um pouco. Como as imagens da galeria se comportam? Elas simplesmente ignoram o tamanho de seu pai `<section class="container` e ficam com muita largura, certo?

Seria interessante se conseguíssemos colocar um comportamento responsivo nas imagens, isto é, que fossem se adaptando de acordo com o elemento-pai delas.

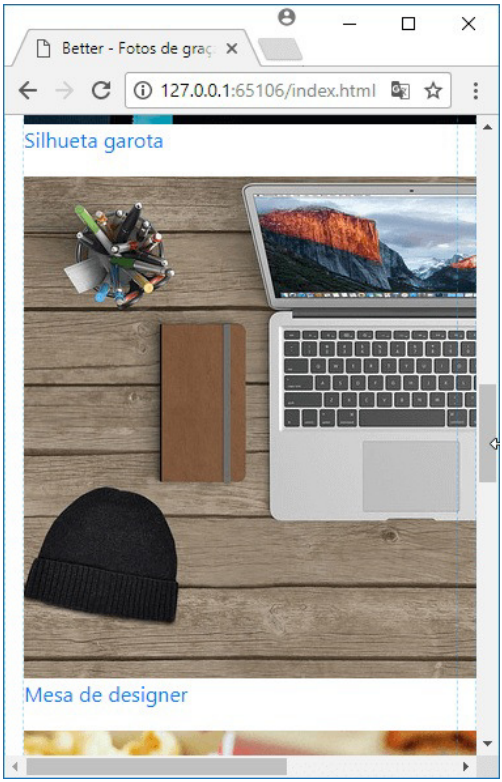


Figura 3.6: Imagens não responsivas

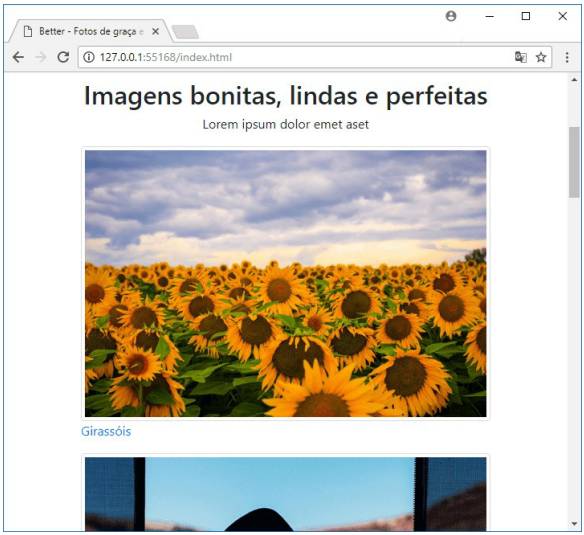
Se você já desenvolveu do lado do *front-end*, deve ter imaginado colocar algumas declarações CSS com *media queries*. Porém, nosso framework preferido pode nos ajudar nisso. Podemos usar a classe `img-fluid` para este fim. E para deixar uma moldura bacana com uma ligeira borda arredondada, podemos usar a classe `img-thumbnail` da seguinte forma:

```
<a href="#">
  <figure>
    <img class="img-fluid img-thumbnail" src="imagens/girassois.j
pg" alt="Girassóis no campo">
```

```
    <figcaption>
      Girassóis
    </figcaption>
  </figure>
</a>
```

> **IMAGENS EM SVG NO IE 10**
>
> Imagens em SVG com a classe `img-fluid` no IE 10 ficam com tamanhos desproporcionais. Uma solução de contorno para isso é adicionar a declaração `width: 100% \9` nessas imagens. Entretanto, tome cuidado, pois isso acabará afetando imagens em outros formatos, como `.jpg` e `.png`, por exemplo, já que estaríamos sobrescrevendo a largura de todas as imagens.

Colocando em todas as imagens, perceba que elas, além de ficarem responsivas, estão com uma bela moldura. Teste redimensionar o browser para vê-las diminuindo.



Perceba também que, apesar de praticamente não mexermos em nada de CSS "na mão", conseguimos fazer uma galeria que, pelo menos, no *mobile*, já se comporta de uma maneira satisfatória. Mas e em tamanhos de tela maiores, como a de um desktop comum, o que acontecerá?

3.5 UM SITE RESPONSIVO PARA TELAS MAIORES

O Bootstrap nasceu como um framework que segue a filosofia de ***mobile first***, ou seja, que leva em consideração se os componentes se comportam bem primeiramente em telas de tamanho reduzido. Pensar em telas médias e grandes, como de desktops, fica para uma segunda instância.

Nosso projeto está bom no mobile, então, vamos prepará-lo

para ficar da mesma forma no desktop também. Nessa galeria, podemos notar que o layout deve se dividir em três colunas, como mostrado no início deste capítulo:

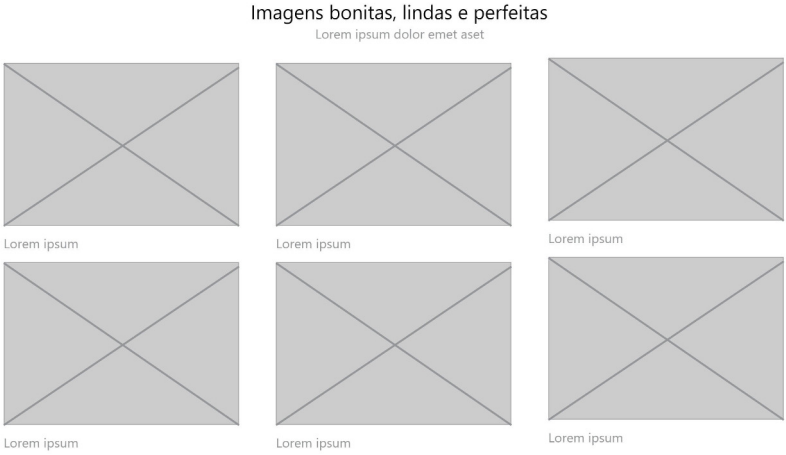


Figura 3.8: Colunas

Como deixamos o layout harmônico visualmente? Usando *grids*! E qual seria um bom número de colunas para essa *grid*: 20, 30, 5? Segundo estudos de design, 12 colunas deixam nossos layouts muito mais flexíveis, por conseguirmos quebrar mais facilmente em outros tamanhos, como 6, 4, 3 ou 2 colunas. É pensando nisso que o *grid* do Bootstrap trabalha com **12 colunas** também.

PARA SABER MAIS SOBRE GRIDS

Se você quer se aprofundar no fantástico mundo das *grids*, recomendo dar uma olhada no livro *Grid Systems*, de Josef Muller Brockmann. Mesmo sendo focado em design editorial, é uma ótima pedida para sair um pouco da nossa bolha de código do dia a dia.

Se as imagens devem ser organizadas de três em três, podemos supor que cada uma delas ocupará 1/3 do tamanho total de sua linha. Para isso, vamos colocar em cada `<a>` de nossa galeria a classe para representar 1/3 de 12 colunas, a `col-4` :

```
  <a href="#" class="col-4">
    <figure>
      <img class="img-fluid img-thumbnail" src="img/garota-silhue
ta.jpg" alt="Silhueta garota">
      <figcaption>
        Silhueta garota
      </figcaption>
    </figure>
  </a>

  <!-- Outros 5 <a> com a classe col-4 -->
```

Classes novas do Bootstrap 4

Se você já trabalhou com Bootstrap antes, saiba que hoje, com a versão 4, a classe usada anteriormente tem como foco dispositivos de tela pequena, o antigo XS (*eXtra Small*) do sistema de grid.

Mas estamos mirando em qual tamanho de tela? Além de dizer a quantidade de colunas, é necessário informar para qual tamanho de tela queremos aquele padrão que estamos aplicando. Como queremos um tamanho de desktop, podemos usar o tamanho que o Bootstrap considera médio, alterando as classes colocadas de `col-4` para `col-md-4`:

```
<a class="col-md-4">
  ...
</a>
```

O QUE É MÉDIO?

Faremos isso para outros tamanhos de tela ainda neste capítulo. Mas, caso você queira um spoiler, veja a figura a seguir:

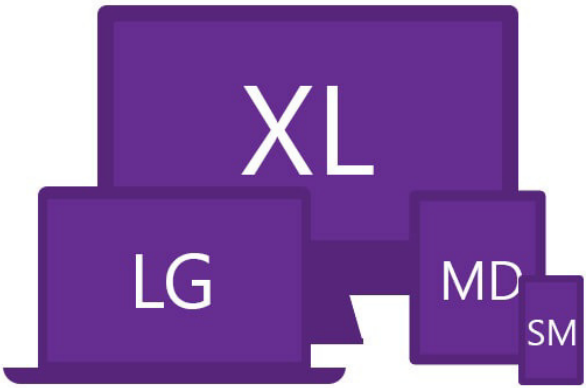


Figura 3.9: Colunas e seus tamanhos

Por fim, sempre que desejarmos utilizar as classes de colunas do framework, é necessário dizer onde começa seu sistema de *grids* usando um elemento de grupo, como uma `<div>`, e contendo a classe responsável por permitir o uso das *grids*, a `.row` (de "linha", em inglês). Então, é só colocar isso em volta dessas colunas. Uma linha será igual a 12 novas colunas disponíveis.

```
<div class="row">
  <a href="#" class="col-md-4">
    ...
  </a>
```

```
  <!-- Outros 5 <a> com a classe col-md-4 -->
</div>
```

Conferindo o resultado no browser, nossa galeria já está organizada e de acordo com o *wireframe*.

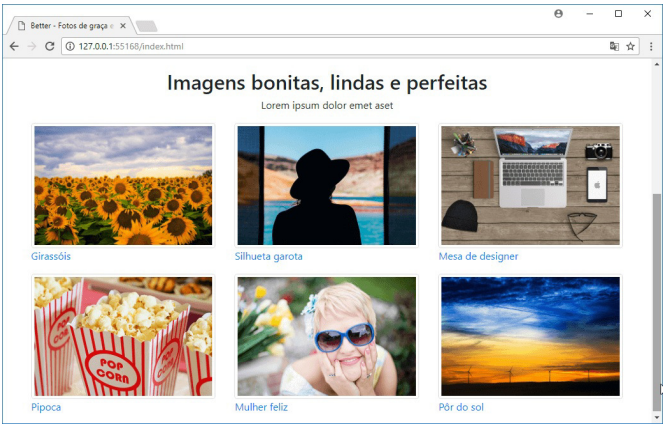


Figura 3.10: Galeria

Vejamos na figura seguinte o que acabamos de fazer:

Figura 3.11: Colunas da galeria

Mas quantas colunas cabem em uma `.row` mesmo? E como que o Bootstrap lida com isso? Simples, cabem apenas doze colunas; mais que isso ele joga para baixo. Foi uma forma inteligente de montar o framework ao pensarmos justamente em

nunca quebrar a *grid*.

Usamos as doze colunas disponíveis na `.row` da *grid* nos três primeiros `<a>`. Agora, se tivermos mais imagens, bastaria acrescentá-las seguindo a mesma estrutura, sempre dentro da linha.

Tudo isso foi feito pensando em tamanhos de tela médios. Mas como fica no mobile? Diminua a tela e veja que, em determinado momento, as colunas começam a empilhar. O que faz todo sentido, já que pensamos em telas reduzidas, pois, se mantivéssemos o mesmo layout de três colunas nesse tamanho de tela, tudo ficaria também com um aspecto apertado e sem respiro.

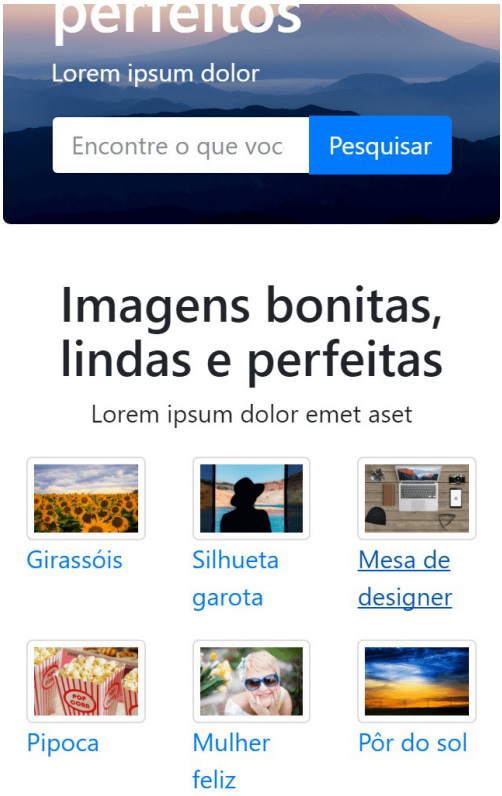


Figura 3.12: Nada interessante

3.6 AJUSTANDO PARA TELAS MAIORES

Nosso layout está bom tanto no mobile como em tamanhos de tela médias. Mas como fazemos para ajustar o *grid* de forma ligeiramente diferente em tamanhos de tela maiores? Como fazer para que as colunas menores, em telas grandes, fiquem como na figura a seguir?

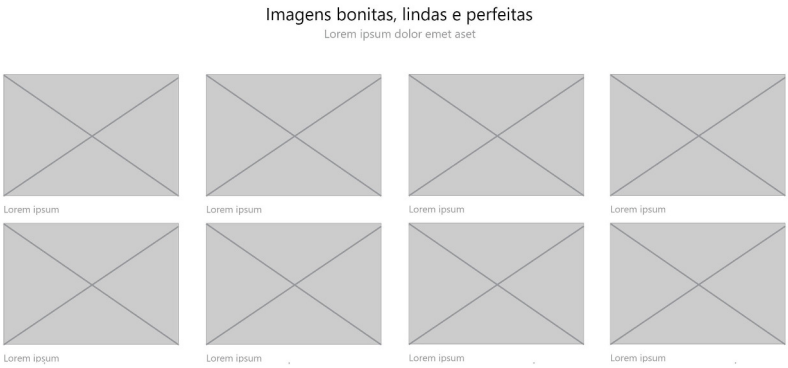
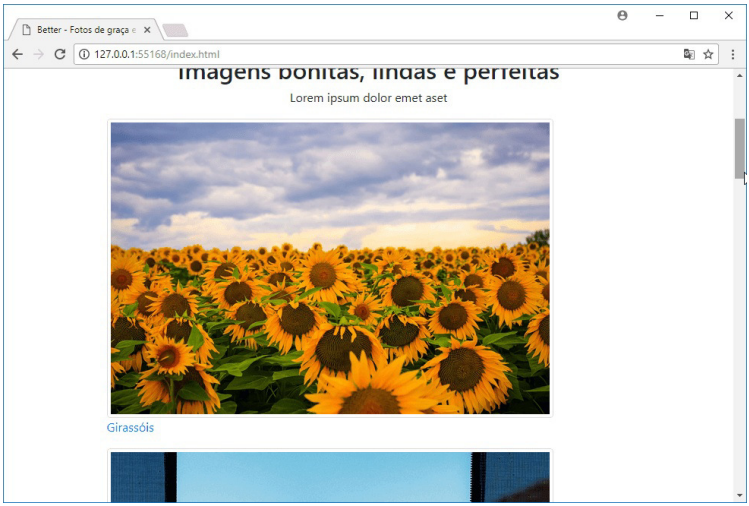


Figura 3.13: Wireframe da galeria com 8 produtos em 4 colunas

Primeiramente, vamos acrescentar mais duas imagens no código, ficando com oito imagens no total, uma embaixo da outra. Além disso, basta alterarmos as classes em nosso código para `col-lg-3` , desta forma:

```
<div class="row">
  <a href="#" class="col-lg-3">
    ...
  </a>

  <!-- Outros 7 <a> com a classe col-lg-3 -->
</div>
```

Com essa alteração, nosso layout estará bom para tamanhos de tela grande. Porém, experimente diminuir um pouco a tela de seu browser para ver que as imagens estão grandes demais em telas médias ou pequenas, assim como antes de aplicarmos as *grids* em nosso projeto!

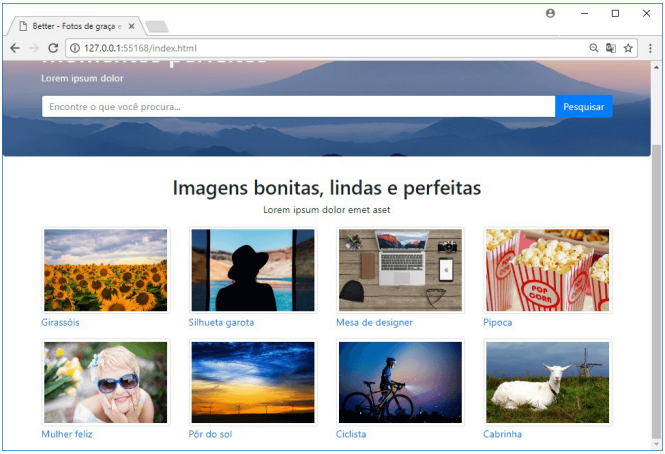


Se deixarmos nosso código da forma como está agora, o layout mobile será aplicado em tamanhos de tela médias para baixo. E qual a razão disso? Lembre-se de que o Bootstrap, desde a versão 3, é *mobile first*, logo, tamanhos de telas pequenas sempre serão priorizados pelo framework. Agora, devemos ajustá-lo para ficar com três colunas em telas médias.

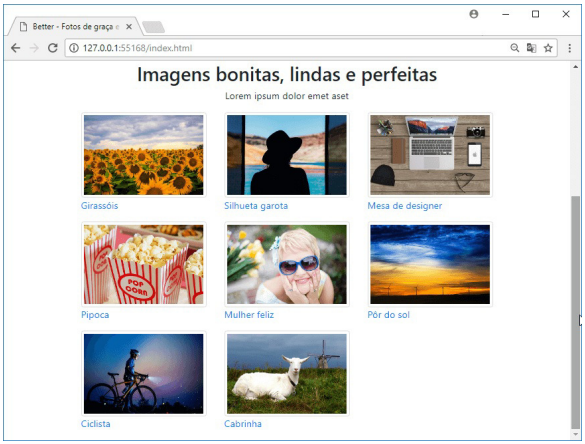
Não podemos aplicar diversas classes no mesmo elemento HTML? E não precisamos aplicar proporções de *grid* diferentes em nosso projeto? Logo, é possível aplicar classes de *grids* diferentes no mesmo elemento HTML:

```
<div class="row">
  <a href="#" class="col-md-4 col-lg-3">
    ...
  </a>

  <!-- Outros 7 <a> com as classes col-md-4 e col-lg-3 -->
</div>
```

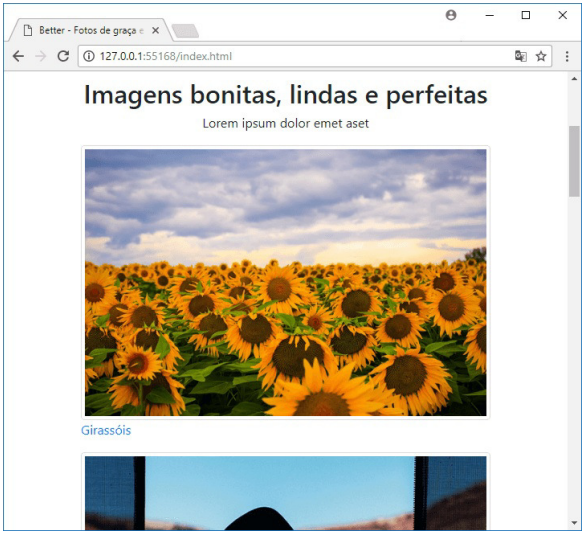
É como se estivéssemos falando que, em telas médias, os <a> precisam ocupar quatro colunas e, em telas grandes, três. Maximize a tela de seu browser (se tiver monitor pequeno, tire um pouco do zoom) para ver o resultado. Agora, tudo estará certo nas telas grandes:



Redimensione a tela do browser para ver como ficou em telas médias:



Podemos não parar por aí! Que tal deixar nosso layout um pouco mais agradável para as telas pequenas? Repare que, quando diminuímos a largura do browser ainda mais, a imagem parece muito grande, cabendo duas imagens lado a lado.



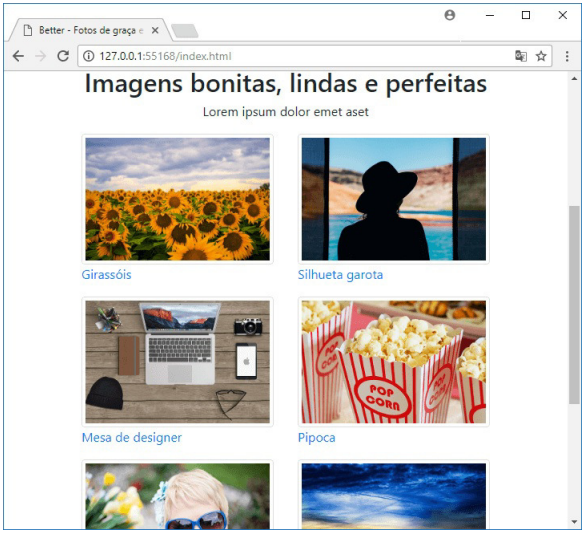
Se o leitor pegou o esquema das colunas, já deve saber o que precisamos fazer. Vamos colocar outra classe específica para tamanhos de tela pequena. Se a linha possui doze colunas e queremos que cada `<div>` ocupe exatamente metade de sua largura total, basta adicionarmos a classe `col-sm-6` nas `<div>`:

```
<a href="#" class="col-sm-6 col-md-4 col-lg-3">
  ...
</a>
```

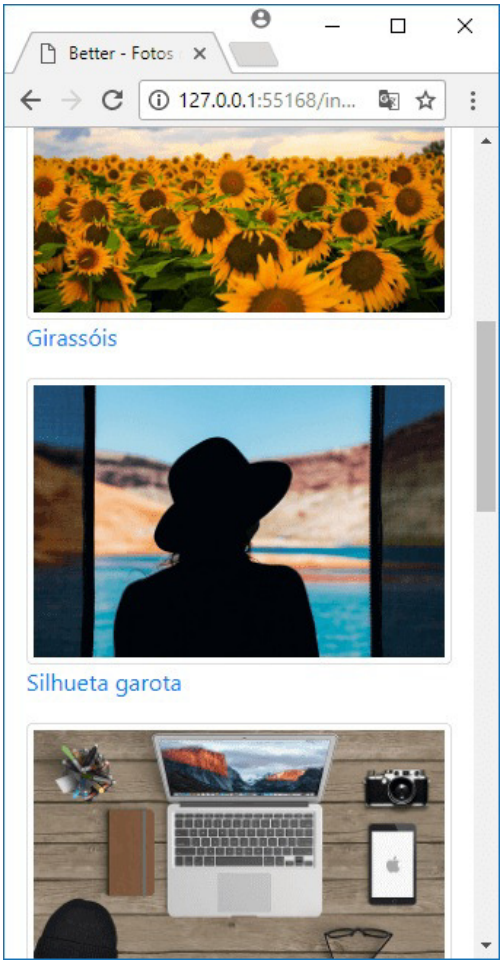
MUITAS CLASSES EM UM MESMO ELEMENTO

Um dos pontos que muitos não gostam do Bootstrap é o uso excessivo de classes nos elementos. Trataremos isso em capítulos posteriores para continuarmos amando este framework em paz.

Diminuindo a tela do browser, podemos ver que está tudo bem agora:



Inclusive em telas extra pequenas:



3.7 RESUMO

Nesta parte do livro, construímos um pouco mais da Better! Agora, seu usuário tem uma ideia da qualidade das imagens que encontrará ali.

Começamos marcando o conteúdo com HTML5 semântico, de

acordo com o wireframe. Também alteramos o `jumbotron`, ao colocá-lo em uma seção dentro do `<main>` e usar a `container` para centralizar tudo, da mesma forma que fizemos no capítulo anterior.

Finalmente começamos a pensar em telas maiores que as de celular, aplicando as famosas *grids* do Bootstrap. Com o intuito de deixar um determinado projeto responsivo, alguns desenvolvedores arrancam apenas essa parte de *grid* do framework. Se seus olhos brilharam com essa possibilidade, fique tranquilo, veremos em alguns capítulos como fazer este procedimento de pegar apenas uma parte do framework.

Ainda falando da *grid*, vimos que o Bootstrap trabalha com uma *grid* de doze colunas, por padrão, a fim de desenvolver layouts mais harmônicos. Claro que, para isso, vimos que é necessário usar a classe `row` e as classes das colunas em si, como `col-md-4` ou `col-lg-3`.

É interessante frisar que também conseguimos com a utilização das classes de colunas adaptar nosso layout original para ficar mais agradável em tamanhos de telas diferentes, apenas usando várias classes de colunas no mesmo elemento.

No próximo capítulo, faremos a próxima seção do site, o **hall da fama**. Vamos adicionar novas regras no nosso tema e, ainda, aplicar *grids* em nosso código.

Capítulo 4

FLEXBOX, AGORA NO BOOTSTRAP

Quem nunca se matou com algum *float* que atire a primeira pedra. Organizar elementos na tela com código pode não ser a coisa mais trivial do mundo à primeira vista. Qual é o motivo de não termos um `float: top` ou um `float: center`, por exemplo? Este tipo de pergunta é bem comum quando estamos começando a aprender CSS. Depois de um tempo, descobrimos que o `float` foi criado para simular o texto em volta de uma imagem e, claro, acabamos pegando essa propriedade para fazer layout.

O próprio Bootstrap sempre trabalhou com o `float` por debaixo dos planos em seu sistema de *grid*, mas, de uns anos para cá, surgiram outros valores CSS e outras propriedades – estas mais específicas para layout, e não uma solução que criamos para suprir nossa demanda por colunas.

Uma dessas fantásticas propriedades é o *flexbox*, ou `display: flex`. Agora, em vez de nos matarmos ajustando individualmente cada elemento na tela, basta aplicar o *flex* ao **elemento-pai** deles e fazer alguns rápidos ajustes. Parece interessante, não?

Não precisamos controlar várias crianças, apenas o pai! Isso nos deixa mais livres para dar uma manutenção mais precisa e ágil em nosso código sempre que surgir alguma alteração, inclusive para tratar a sua responsividade.

Vamos brincar um pouco com o *flexbox* nesta parte do livro e, claro, usar o Bootstrap como base dessa brincadeira.

Aquela olhada básica na planta da casa

Como de costume, vamos olhar o *wireframe* do projeto:

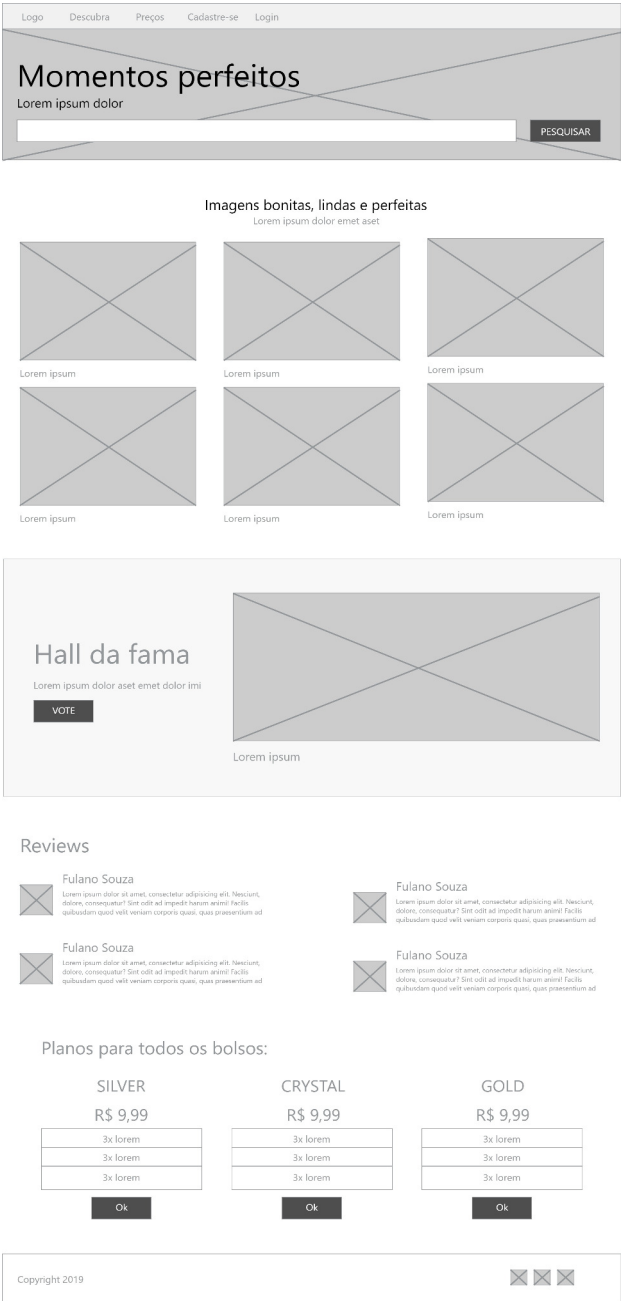

Logo
Descubra
Preços
Cadastre-se
Login
Momentos perfeitos
Lorem ipsum dolor
PESQUISAR
Imagens bonitas, lindas e perfeitas
Lorem ipsum dolor emet aset
Lorem ipsum
Lorem ipsum
Lorem ipsum
Lorem ipsum
Lorem ipsum
Lorem ipsum
Hall da fama
Lorem ipsum dolor aset emet dolor imi
VOTE
Lorem ipsum
Reviews
Fulano Souza
Fulano Souza
Fulano Souza
Fulano Souza
Planos para todos os bolsos:
SILVER
R$ 9,99
3x lorem
3x lorem
3x lorem
Ok
CRYSTAL
R$ 9,99
3x lorem
3x lorem
3x lorem
Ok
GOLD
R$ 9,99
3x lorem
3x lorem
3x lorem
Ok
Copyright 2019

Figura 4.1: Wireframe do Hall da Fama

Com o conhecimento adquirido no capítulo anterior sobre as *grids*, podemos notar que temos duas grandes colunas, uma com o título e parágrafo e, na outra, a imagem. Vamos começar a brincadeira!

4.1 NÃO É TV FAMA, É HALL DA FAMA!

Muito bom, já conhecemos o processo inicial para criarmos uma nova seção da Better. Primeiramente, vamos apenas colocar o conteúdo que precisamos marcar como consta no *wireframe*.

```
  Hall da fama do mês de janeiro

  Por votação do público e número de downloads,
  escolhemos todo mês as fotografias mais baixadas
  do mês anterior e disponibilizamos de graça.

    Quero votar!


    Girassóis
```

Outra parte do site, logo, outra seção do site. E apenas para identificação, vou identificá-la com uma classe que demonstre do que este bloco se trata:

```
<section class="hall">
  Hall da fama do mês de janeiro

  Por votação do público e número de downloads,
  escolhemos todo mês as fotografias mais baixadas
  do mês anterior e disponibilizamos de graça.

    Quero votar!
```

```
    Girassóis
</section>
```

Além disso, vamos usar elementos comuns do HTML para cada bloco de informação, como `<h1>` e `<p>`, da mesma forma que fizemos nos capítulos anteriores. Vale notar também que teremos um botão para que o usuário vote na fotografia que ele acha que deve ir para o Hall da Fama da Better. E, ao lado, teremos uma imagem com sua respectiva legenda:

```
<section class="hall">
  <h1>
    Hall da fama do mês de janeiro
  </h1>

    <p>
        Por votação do público e número de downloads,
        escolhemos todo mês as fotografias mais baixadas
        do mês anterior e disponibilizamos de graça.
    </p>

    <button>
        Quero votar!
    </button>

    <figure>
        <img src="imagens/girassois.jpg" alt="Girassóis no campo"
>
        <figcaption>
              Girassóis
        </figcaption>
    </figure>

</section>
```

Vendo o resultado no browser, não temos nada de inesperado, pois ainda não aplicamos nada de estilo:

Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!

Figura 4.2: Hall sem classe

Vamos começar atacando o botão. Como vimos em capítulos anteriores, o Bootstrap possui classes para formatarmos um elemento como um botão, além de conseguirmos deixá-lo grande e com a cor primária usada pelo framework. Vamos aplicá-las agora, tendo como novidade a classe `btn-lg`, que deixa o botão com um tamanho maior que o padrão, coloca mais *padding* e aumenta um pouco sua fonte:

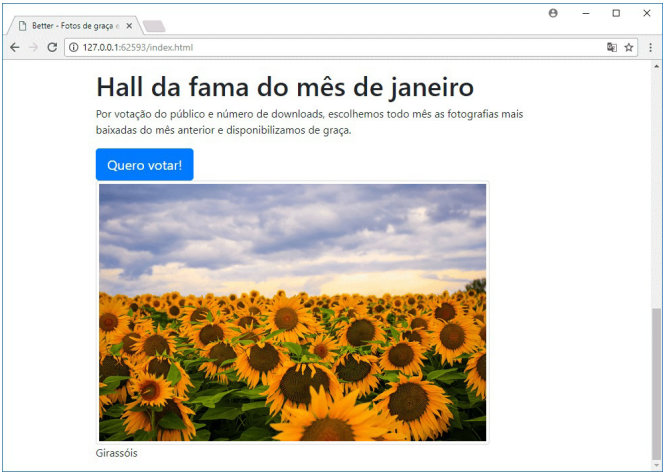
```
<section class="hall">
  <h1>
    ...
  </h1>

    <p>
      ...
    </p>

    <button class="btn btn-lg btn-primary">
        Quero votar!
    </button>

    <figure>
        ...
    </figure>

</section>
```

Além disso, podemos usar a classe `container` para

centralizar o conteúdo dessa seção inteira:

```
<section class="hall">
    <div class="container">
        ...
    </div>
</section>
```

Ao visualizar no browser, vemos que estamos progredindo:


Better - Fotos de graça
127.0.0.1:62593/index.html
Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!
Girassóis

Você se lembra da classe `img-thumbnail`, que usamos nas imagens da galeria? Vamos usá-la novamente a fim de deixar aquela leve borda arredondada na imagem. Além disso, vamos utilizar a classe responsável por deixar a imagem responsiva, a `img-fluid`:

```
<section class="hall">
  <div class="container">
    ...

    <figure>
      <img class="img-fluid img-thumbnail" src="img/girassois.jpg
" alt="Girassóis no campo">
```

```
      <figcaption>
        Girassóis
      </figcaption>
    </figure>
  </div>
</section>
```

Ou seja, o que foi feito foi basicamente igual ao que fizemos nas imagens da galeria. Então, vamos começar a aplicar a *grid* do Bootstrap ao nosso Hall da Fama. Primeiramente, o que tínhamos de fazer para ter acesso às doze colunas? Fazer uma linha! Assim, coloquemos uma `<div>` com a classe `.row` e, para dar uma organizada, vamos comentar a *tag* de fechamento da `.container`:

```
<section class="hall">
  <div class="container">
    <div class="row">

      <!-- H1, P, BUTTON, FIGURE -->

    </div>
  </div> <!-- fim .container hall -->
</section>
```

Agora, basta separarmos os dois grandes blocos de conteúdo que vimos no *wireframe* em colunas. Digamos que estamos mirando em telas grandes e que queremos que cada parte tenha exatamente metade da `.row`, então, quantas colunas teremos para cada? Seis! Para fazer isso, criaremos duas `<div>` com a classe `col-lg-6` em cada uma delas:

```
<section class="hall">
  <div class="container">
    <div class="row">
      <div class="col-lg-6">
        <!-- H1, P, BUTTON -->
      </div>
```

```
      <div class="col-lg-6">
        <!-- FIGURE -->
      </div>

    </div> <!-- fim .row hall-->>
  </div> <!-- fim .container hall -->
</section>
```

IGUAL AO WIREFRAME

Como você faria para deixar essa divisão com outra proporção? Lembre-se de que você pode brincar com os valores colocados nas colunas para deixar o layout da forma que mais lhe agrade!

Uma pergunta: qual é a função dessa `<figure>` ? Ela é um elemento semântico para empacotar a imagem e a legenda, ou seja, torná-las filhas de um mesmo pai – assim como as `<div>` que criamos por todo o nosso código.

Relaxe que está tudo funcionando agora, mas vamos focar em deixar nosso HTML o mais limpo possível. Tiraremos a segunda `<div class="col-lg-6"` e colocaremos a classe de coluna diretamente na `<figure>` :

```
<section class="hall">
  <div class="container">
    <div class="row">

      <div class="col-lg-6">
        <!-- H1, P, BUTTON -->
      </div>

      <figure class="col-lg-6">
        ...
```

```
      </figure>

    </div> <!-- fim .row hall-->>
  </div> <!-- fim .container hall -->
</section>
```

Testando no browser, tudo está bem por enquanto:

Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!
Girassóis

Figura 4.4: Hall com colunas

4.2 O FLEXBOX MAIS RÁPIDO DO OESTE

Agora precisamos centralizar verticalmente a coluna da esquerda com a da direita (a imagem), mas como fazemos isso?

Na versão 3 do Bootstrap, precisaríamos criar regras específicas para esta parte do nosso código. E isto envolveria mudar a forma de exibição das colunas para uma exibição baseada em tabelas, isto é, uma bela de uma solução de contorno!

Ainda no Bootstrap v3?

Acesse no link a segui para ver a solução citada anteriormente, caso você esteja mexendo com a versão 3 do Bootstrap ou tenha curiosidade: https://codepen.io/designernatan/pen/mBYGdw.

Centralizar elementos sempre foi um pequeno desafio, mesmo para *front-enders* mais experientes, pois são muitas variantes, como altura do elemento, forma de exibição do elemento-pai etc. Este trabalho ficou mais simples com o advento do **flexbox**, uma forma relativamente nova e bem interessante para organizar elementos pelo CSS, em que controlamos apenas o pai dos elementos que queremos mexer.

É uma forma muito poderosa e eficiente de criar *layouts* em sites. Recomendo que, se você quiser conhecer um pouco sobre o *flexbox*, dê uma olhada neste jogo que ensina, de uma maneira fantástica, como aplicar as principais funcionalidades do *flex*: http://flexboxfroggy.com.

A fim de não nos preocuparmos mais com browsers antigos e problemas de layout, e termos mais flexibilidade para diagramar nossas interfaces, o Bootstrap 4 optou por usar o *flexbox* por baixo dos panos como o principal motor de seu sistema de *grid*. Ele abandonou o `float` e outras propriedades que foram usadas desde a primeira versão do framework.

Como dito, precisamos controlar o pai dos elementos que

estamos mirando. No caso, quem é o pai direto das colunas do hall? A `.row`! Vamos fazer isso apenas **adicionando** a classe `d-flex` a ela, a fim de ligarmos o modo *flexbox* em todo esse elemento:

```
<div class="row d-flex">
  <div class="col-lg-6">
    ...
  </div>
  <figure class="col-lg-6">
    ...
  </figure>
</div>
```

Ao testarmos, não temos nada. De certa forma, acabamos de habilitar o recurso do *flex* nessa `.row`. Ainda precisamos centralizar seu conteúdo verticalmente. Para isso, vamos adicionar na mesma `.row` a classe para alinhar itens ao centro (`align-items-center`):

```
<div class="row d-flex align-items-center">
  <div class="col-lg-6">
    ...
  </div>
  <figure class="col-lg-6">
    ...
  </figure>
</div>
```

Testando no browser, conseguimos centralizar verticalmente todo o conteúdo da primeira coluna!

Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!
Girassóis

Figura 4.5: Hall ficando bacana

JUSTIFICANDO LINHAS E COLUNAS

Recomendo dar uma olhada rápida nas possibilidades de alinhamento que o *flexbox* nos permite usando o Bootstrap: https://getbootstrap.com/docs/4.0/utilities/flex/.

4.3 AJEITANDO O HALL

Conseguimos fazer a seção do *hall* de uma maneira rápida com o Bootstrap. Porém, repare no wireframe novamente, no início do capítulo, e veja que existe uma cor no fundo dessa seção.

Com sua versão 4, o Bootstrap facilitou nossa vida até em situações simples como esta. Para colocar um *background* cinza escuro, basta adicionarmos a classe `.bg-dark` na seção relativa ao *hall*:

```
<section class="hall bg-dark">
  <div class="container">
    <div class="row">
```

```
      ...
    </div>
  </div>
</section>
```

Olhando no browser, podemos notar que nem tudo são flores nessa vida:

Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!
Girassóis

Figura 4.6: Hall escuro

Toda vez que você deixar um fundo escuro com texto escuro em cima, saiba que um filhote de cabra tropeça e bate a cabeça no chão com bastante força. Cadê a acessibilidade, jovem Padawan? Vamos deixar o título, o parágrafo e a legenda da figura mais claros!

5 DICAS PARA MELHORAR A ACESSIBILIDADE EM SUA INTERFACE

Quer aprender mais sobre acessibilidade? Dá uma lida neste post gigante que fiz sobre o tema: http://blog.caelum.com.br/melhorando-a-acessibilidade-em-suas-interfaces.

Esta é outra situação em que o Bootstrap nos ajuda de forma

eficiente, possuindo classes para mudarmos as cores do texto. No nosso caso, precisamos de um texto mais claro, então vamos usar a classe `.text-light`, que deixa o texto em uma cor parecida com o branco! Mas em quem faremos isso? Nos três elementos mencionados? Seria muito mais prático pegar a seção inteira, uma vez que todos os textos seguirão o mesmo estilo:

```
<section class="hall bg-dark text-light">
  <div class="container">
    <div class="row">
      ...
    </div>
  </div>
</section>
```

Outras cores

Sim, existem outras classes do Bootstrap responsáveis por aplicar diferentes cores no fundo e nos textos dos elementos. Dê uma olhada nas possibilidades: https://getbootstrap.com/docs/4.0/utilities/colors.

Vamos conferir o sucesso no browser:

Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!
Girassóis

Figura 4.7: Hall escuro

O último detalhe agora seria alinhar a imagem ao centro, como fizemos com a coluna da esquerda. Uma forma possível para se fazer isso seria aumentar o espaço interno da seção do *hall*, ou seja, aumentar seu *padding*.

"Ah, Natan, é impossível o Bootstrap ter classes para isso também!". Pois é, ele tem! Nosso framework do coração possui diversas classes utilitárias em formato de atalho de margens, e *paddings* para modificar a aparência dos elementos e seus **espaçamentos**.

As classes de espaçamento do Bootstrap funcionam de uma forma bem didática. Se queremos aplicar um `padding` no topo para todos os tamanhos de tela, da pequena a grande, na seção *hall*, precisamos aplicar a classe `.pt-sm-X` (repare que `pt` significa `padding-top`). E o que seria o X? É o quão grande será este espaço!

No nosso caso, um espaço razoavelmente grande ficaria visualmente agradável, então, vamos pegar de um total de apenas cinco, isto é, quase o maior valor possível que o framework nos dá. Deixaremos nosso código assim:

```
<section class="hall bg-dark text-light pt-sm-4">
  ...
</section>
```

Para aplicarmos a **todos** os tamanhos de tela de acordo com o sistema de *grid*, podemos alterar a classe para `.pt-4`:

```
<section class="hall bg-dark text-light pt-4">
  ...
</section>
```

Agora é só curtir o resultado no browser:

Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!
Girassóis

De onde veio esse número 4?

O *range* dos valores que vão de 0 a 5 nessas classes de espaçamento é relacionado ao tamanho da fonte do HTML, a unidade `rem`. Quando colocamos `.pt-1`, estamos colocando um `padding-top` de 0.25rem. Se fosse **2** ali no valor, seria 0.5rem, e assim até chegarmos ao valor 5, que corresponde a 3rem.

Você pode conferir a documentação das classes de espaçamento aqui: https://getbootstrap.com/docs/4.0/utilities/spacing/.

Diminuindo a tela do browser, podemos notar outro problema de espaçamento: a figura está muito colada ao botão. Como poderíamos resolver isso?

Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!

Seguindo a mesma estratégia que acabamos de usar, utilizaremos as classes de espaçamento para nos ajudar. Vamos criar uma margem superior (`margin-top`) à figura, assim, adicionaremos a classe `mt-4` no elemento `<figure>` :

```
<figure class="col-lg-6 mt-4">
  ...
</figure>
```

Teste no browser, com a tela pequena, e veja quão maravilhoso ficou nosso *hall* da fama:

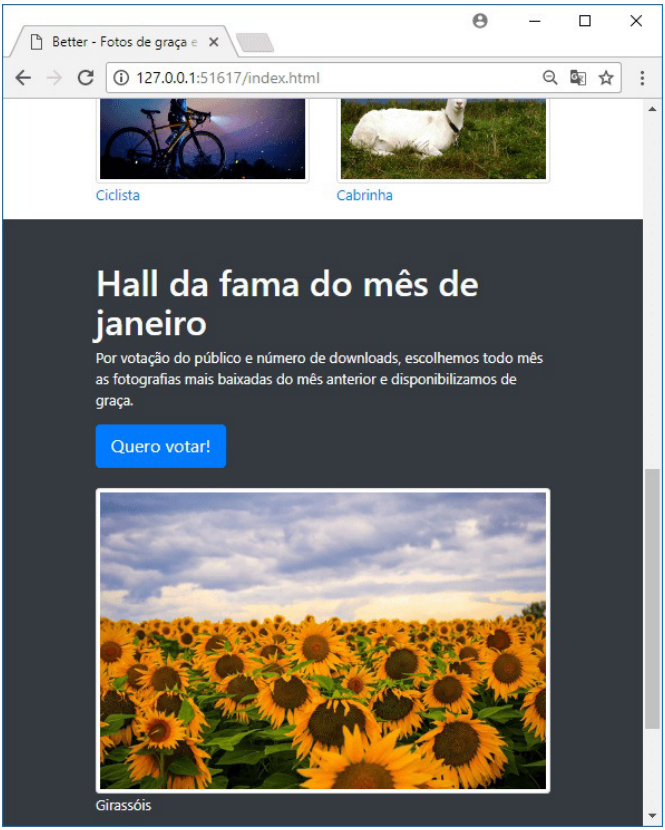

Better - Fotos de graça e
127.0.0.1:51617/index.html
Ciclista
Cabrinha
Hall da fama do mês de janeiro
Por votação do público e número de downloads, escolhemos todo mês as fotografias mais baixadas do mês anterior e disponibilizamos de graça.
Quero votar!
Girassóis

Figura 4.10: Hall ok

4.4 RESUMO

Como sempre, vamos ver rapidamente o que aprendemos neste capítulo. Vimos que, ao marcar o conteúdo, devemos nos preocupar com a semântica de nosso código. Isso resume-se a utilizar os elementos do HTML5 de forma coerente e fazendo sentido com o conteúdo que estamos marcando.

Apesar de o Bootstrap ter na documentação vários tipos de

`<div>`, quase que infinitos, aquilo não passa de uma **recomendação**. Se será usado uma `<div>` ou uma `<figure>`, cabe a nós, desenvolvedores, decidir quais elementos utilizar.

Tivemos também um panorama geral da forma que o framework organiza sua estrutura de layout por baixo dos panos, sendo que, em sua versão 4, o *flexbox* foi usado pelo seu sistema de *grid* por padrão.

Por fim, customizamos a cor de fundo e textos da seção *hall* com classes utilitárias relacionadas às cores, como `bg-dark` e `text-light`. Também conseguimos ajustar os espaçamentos internos e externos de dois elementos usando as classes `pb-4` e `mt-4`, para alterar respectivamente o `padding-bottom` e a `margin-top`.

No capítulo seguinte, faremos a seção de *feedbacks* dos usuários da Better. Vamos mexer um pouco mais com o *flexbox* do Bootstrap e aprender um componente novo, o *media*.

Capítulo 5

MEDIA, UM COMPONENTE ATUALIZADO

Tecnologias sempre atualizam, e esta é a regra da vida front-end. Elas se inovam, reinventam e reciclam, seja o HTML, o framework JavaScript da moda no **mês**, a IDE mais robusta atualmente etc. Com o Bootstrap, isso, sem dúvida alguma, não seria diferente.

Desde a sua primeira versão, o Bootstrap dava indícios de que se tornaria popular, por facilitar a vida do desenvolvedor, fosse para entender o código dos componentes ou mesmo para achar algum deles em sua documentação. Queira ou não, o Bootstrap foi um marco, da mesma forma que o jQuery. Inclusive, temos *haters* e defensores para ambos.

Home
Get started
Scaffolding
Base CSS
Components
JavaScript
Customize
Bootstrap
Bootstrap
Sleek, intuitive, and powerful front-end framework for faster and easier web development.
Download Bootstrap
GitHub project
Examples
Extend
Version 2.3.2
Follow @getbootstrap
Tweet
Heads up! These docs are for v2.3.2, which is no longer officially supported. Check out the latest version of Bootstrap!
Introducing Bootstrap.
Need reasons to love Bootstrap? Look no further.
By nerds, for nerds.
Built at Twitter by @mdo and @fat, Bootstrap utilizes LESS CSS, is compiled via Node, and is managed through GitHub to help nerds do awesome stuff on the web.
Made for everyone.
Bootstrap was made to not only look and behave great in the latest desktop browsers (as well as IE7!), but in tablet and smartphone browsers via responsive CSS as well.
Packed with features.
A 12-column responsive grid, dozens of components, JavaScript plugins, typography, form controls, and even a web-based Customizer to make Bootstrap your own.

Figura 5.1: Bootstrap em sua segunda versão

Perceba também como o layout do framework evoluiu, acompanhando totalmente as tendências de design de acordo com sua época. Na versão da figura anterior, vemos a presença de gradientes e sombras fortes. Na atual, temos o *flat design* com cores mais chapadas e um uso ponderado de sombras.

Você que mexe com isso desde as versões mais antigas deve se lembrar bem do componente chamado *media*, uma verdadeira mão na roda para quando queríamos evitar dores de cabeça com *floats* na época. Ele ainda existe, mas evoluiu. Com o objetivo principal de praticar mais, vamos ver o que tem de novo por baixo dos panos e "codar" mais uma seção do site.

5.1 UM ANTIGO COMPONENTE

No wireframe do projeto, veja que a seção relacionada aos reviews e feedbacks do usuário possui um título principal e o review em si, que, por sua vez, possui uma imagem, um título e um pequeno parágrafo. Além disso, tudo isso tudo está repetido quatro vezes:

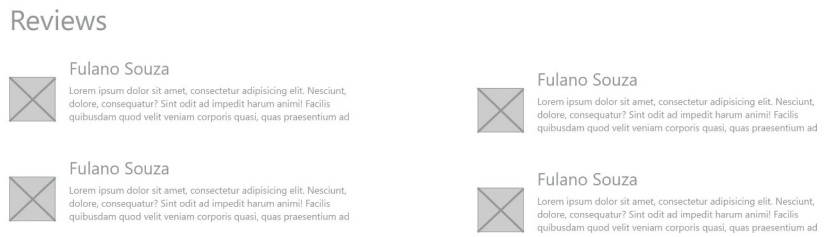


Figura 5.2: Wireframe da seção de reviews

Começando os trabalhos, vamos criar essa nova seção abaixo da seção do *hall*:

```
<section class="hall bg-dark text-light pt-5">
    ...
</section>

<section class="reviews">

</section>
```

Além disso, vamos colocar os conteúdos ali dentro. Teremos quatro *reviews* ao todo, mas, primeiro, faremos um único bloco que representará um único review. Mais para a frente deste capítulo, vamos copiar e colá-lo três vezes:

```
<section class="reviews">
    Reviews

    Roberto Ford

    Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nes
```

```
ciunt, dolore, consequatur? Sint odit ad impedit harum animi! Fac
ilis quibusdam quod velit veniam corporis quasi, quas praesentium
 ad accusantium quaerat consequuntur.
</section>
```

Confira no *wireframe* que precisamos colocar uma imagem onde será mostrada a foto do usuário que deixou o review. Por hora, vamos colocar uma imagem qualquer (ou imagem coringa):

```
<section class="reviews">
    Reviews

  <img src="https://dummyimage.com/64x64/000000/fff" alt="Roberto
 Ford">

    Roberto Ford

    Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nes
ciunt, dolore, consequatur? Sint odit ad impedit harum animi! Fac
ilis quibusdam quod velit veniam corporis quasi, quas praesentium
 ad accusantium quaerat consequuntur.
</section>
```

Esse texto `"Roberto Ford"` tem o mesmo peso semântico que o texto `"Reviews"`? Eles são só dois blocos de texto perdidos com a mesma importância de significado? Não! Podemos chegar à conclusão de que, para começar a marcação, teremos de respeitar a hierarquia que nos é mostrada no *wireframe*, como título, subtítulo e um parágrafo, usando os elementos `<h1>`, `<h2>` e `<p>`:

```
<section class="reviews">
  <h1>
    Reviews
  </h1>

  <img src="https://dummyimage.com/64x64/000000/fff" alt="Roberto
 Ford">

  <h2>
    Roberto Ford
  </h2>
```

```
  <p>
    Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nes
ciunt, dolore, consequatur? Sint odit ad impedit harum animi! Fac
ilis quibusdam quod velit veniam corporis quasi, quas praesentium
 ad accusantium quaerat consequuntur.
  </p>
</section>
```

Veja no browser que estamos um tanto longe do resultado esperado no final, mas, com carinho, chegaremos lá:

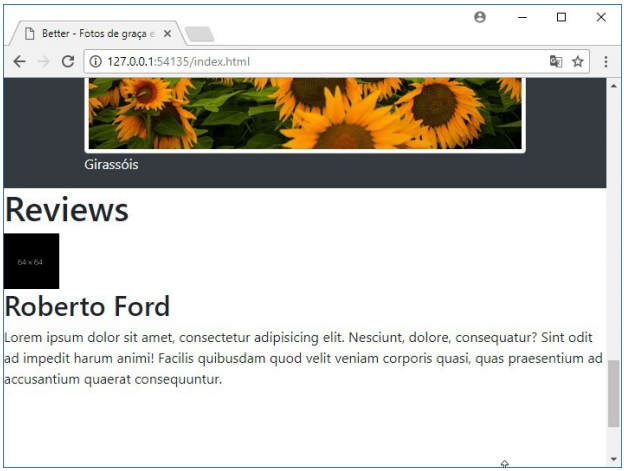

Better - Fotos de graça
127.0.0.1:54135/index.html
Girassóis
Reviews
64 x 64
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.

Figura 5.3: O começo dos reviews

Repare nessa imagem que precisamos repetir esse bloco de imagem, subtítulo e parágrafo quatro vezes. Precisamos primeiramente colocar tudo isso em um grande bloco, assim, vamos usar uma `<div>` para isso. E para o Bootstrap começar a agir deixando a imagem ao lado do título, como se estivéssemos usando o *flexbox*, vamos usar o já citado *media*:

```
<section class="reviews">
  <h1>
    Reviews
```

```
  </h1>

    <div class="media">
      <img src="https://dummyimage.com/64x64/000000/fff" alt="Rob
erto Ford">

      <h2>
        Roberto Ford
      </h2>

      <p>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit.
 Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi!
 Facilis quibusdam quod velit veniam corporis quasi, quas praesen
tium ad accusantium quaerat consequuntur.
      </p>
    </div>
</section>
```

Girassóis
Reviews
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.

Figura 5.4: Conteúdo amontoado de um review

Isso ainda está estranho. Se você já brincou com o *flexbox*, deve ter visto um comportamento de tentar encaixar todos os filhos de um elemento-pai na mesma linha, mesmo que fiquem apertados. Isto é justamente o *flexbox* atuando ali, por baixo dos panos!

A ideia é a imagem ficar à esquerda, e os textos, à direita. Para isso, devemos agrupar os textos em um grande bloco, que representará o corpo principal do nosso elemento *media*. Vamos então criar uma `<div>` com a classe `.media-body`:

```
<section class="reviews">
  <h1>
    Reviews
```

```
  </h1>

  <div class="media">
    <img src="https://dummyimage.com/64x64/000000/fff" alt="Rober
to Ford">

    <div class="media-body">
      <h2>
        Roberto Ford
      </h2>

      <p>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit.
 Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi!
 Facilis quibusdam quod velit veniam corporis quasi, quas praesen
tium ad accusantium quaerat consequuntur.
      </p>
    </div>
  </div>
</section>
```

Veja como aos poucos estamos deixando tudo igual à proposta:

Girassóis
Reviews
Roberto Ford
64 x 64
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.

Figura 5.5: Review com o media-body

Agora, o nome do usuário que deixou o *review* está muito grande. Poderíamos diminuí-lo rapidamente trocando o `<h2>` por um elemento de título de menor importância, como um `<h5>`, por exemplo. Mas como ficaria a semântica do nosso código? Temos um `<h1>` na página e, do nada, pularemos para o nível 5?

Mesmo considerando que, na documentação do elemento

media, usa-se o `<h5>`, sempre devemos nos perguntar se faz realmente sentido a marcação sugerida no GetBootstrap. Teremos um capítulo para discutir, testar e projetar esta ideia, levando em conta este e outros casos.

E agora? Mudaremos o `font-size` na mão? De fato, fazer desta forma é algo fácil e rápido. Mas, e se o Bootstrap fornecesse uma classe para isso, não seria **ainda mais prático?** Pois bem, eles pensaram até nisso quando desenvolveram o framework!

O Bootstrap possui diversas classes utilitárias relacionadas à estilização de tipografias. Se queremos deixar o `<h2>` com o tamanho próximo de um `<h5>`, usaremos a classe `.h5` nele:

```
<section class="reviews">
    <h1>
        Reviews
    </h1>

  <div class="media">
    <img src="https://dummyimage.com/64x64/000000/fff" alt="Rober
to Ford">

    <div class="media-body">
      <h2 class="h5">
        Roberto Ford
      </h2>

      <p>
        ...
      </p>
    </div>
  </div>
</section>
```

Com isso, o `<h2>` ficará menor. Não tivemos de colocar nada manualmente e, acima disso tudo, nosso código se mantém semântico.

Utilitário de tipografia

Ficou curioso para ver quais são as outras possibilidades que o Bootstrap tem em relação à tipografia? Acesse: https://getbootstrap.com/docs/4.0/content/typography/.

Desta forma, podemos usar as classes utilitárias de margem e `padding` que vimos no capítulo anterior para afastar um pouco a imagem do usuário dos textos ao seu lado. Se queremos aplicar uma margem do lado direito (*right*), podemos fazê-lo da seguinte maneira:

```
<section class="reviews">
    <h1>
        Reviews
    </h1>
  <div class="media">
    <img src="https://dummyimage.com/64x64/000000/fff" alt="Rober
to Ford" class="mr-3">

    <div class="media-body">
      ...
    </div>
  </div>
</section>
```

O resultado esperado, agora com o primeiro *review* configurado corretamente, é o seguinte:

Girassóis
Reviews
64 × 64
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.

Figura 5.6: Primeiro review quase ok

Agora, devemos apenas corrigir o alinhamento dessa seção. É possível usar nossa velha e conhecida classe `.container`. Precisaremos criar uma divisão dentro da seção, ou poderemos usar direto na `<section>`?

Atentando-nos à proposta do *wireframe*, como não há nenhum *background* que necessite ser esticado juntamente com a janela, podemos aplicar diretamente na `<section>`:

```
<section class="reviews container">
    <h1>
        Reviews
    </h1>
  <div class="media">
    <img src="https://dummyimage.com/64x64/000000/fff" alt="Rober
to Ford" class="mr-3">

    <div class="media-body">
      ...
    </div>
  </div>
</section>
```

Um detalhe estético: vamos aplicar também uma margem inferior um pouco maior no título principal da seção *reviews*, apenas para deixá-la mais espaçada e com um respiro maior:

```
<section class="reviews container">
    <h1 class="mb-3">
        Reviews
```

```
    </h1>
  <div class="media">
    ...
  </div>
</section>
```

Com isso, nossa seção de *reviews* está centralizada:

Girassóis
Reviews
Roberto Ford
64 x 64
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.

Figura 5.7: Área de reviews agora com container

MAIS SOBRE O MEDIA

Se deseja ver outras possibilidades e usos para o *media*, confira sua documentação: http://getbootstrap.com/docs/4.0/layout/grid/.

Com a nossa `<div class="media">` inteira e multiplicada por quatro, podemos mudar o conteúdo a fim de deixar o layout mais crível:

```
<h1 class="mb-3">
  Reviews
</h1>

<div class="media">
  ...
</div>
<div class="media">
  ...
</div>
```

```
<div class="media">
  ...
</div>
<div class="media">
  ...
</div>
```

Testando no browser, vemos que fica tudo esticado, tanto no mobile quanto no desktop:

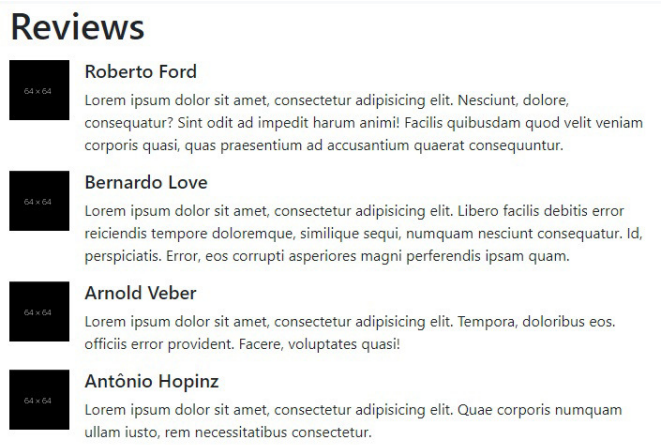

Reviews
64 x 64
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.
64 x 64
Bernardo Love
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero facilis debitis error reiciendis tempore doloremque, similique sequi, numquam nesciunt consequatur. Id, perspiciatis. Error, eos corrupti asperiores magni perferendis ipsam quam.
64 x 64
Arnold Veber
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempora, doloribus eos. officiis error provident. Facere, voluptates quasi!
64 x 64
Antônio Hopinz
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quae corporis numquam ullam iusto, rem necessitatibus consectetur.

Figura 5.8: Reviews ok, mas ainda esticados

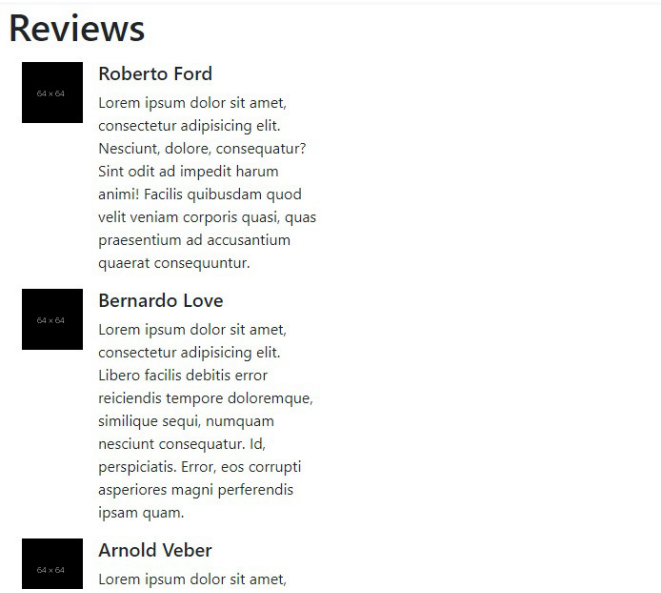
Qual recurso do Bootstrap podemos usar para ajudar na diagramação dessa seção? As *grids*! Como nosso layout precisa ser *meio a meio*, bastando adicionar a classe que corresponde à metade do número de colunas do sistema de *grid* do Bootstrap (no caso, estamos usando doze)!

Sempre com *mobile first* em mente, vamos inicialmente mirar no menor tamanho de tela possível, o antigo *eXtra Small*. Lembre-se de que basta colocar `col-x`, em que `x` é o número de colunas:

```
<h1 class="mb-3">
```

```
  Reviews
</h1>

<div class="media col-6">
  ...
</div>
<div class="media col-6">
  ...
</div>
<div class="media col-6">
  ...
</div>
<div class="media col-6">
  ...
</div>
```

Abra seu browser e veja que não há mais nada esticado. Contudo, as colunas não estão se encaixando uma ao lado da outra:


Reviews
64 × 64
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.
64 × 64
Bernardo Love
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero facilis debitis error reiciendis tempore doloremque, similique sequi, numquam nesciunt consequatur. Id, perspiciatis. Error, eos corrupti asperiores magni perferendis ipsam quam.
64 × 64
Arnold Veber
Lorem ipsum dolor sit amet,

Sempre que usamos as *grids*, devemos lembrar de primeiro criar uma linha, como vimos nos capítulos anteriores. Para isso, vamos utilizar a classe `row` :

```
<h1 class="mb-3">
  Reviews
</h1>
<div class="row">
  <div class="media col-6">
    ...
  </div>
  <div class="media col-6">
    ...
  </div>
  <div class="media col-6">
    ...
  </div>
  <div class="media col-6">
    ...
  </div>
</div>
```

Testando no browser, veremos que tudo está certo:

Reviews
64 x 64
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.
64 x 64
Bernardo Love
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero facilis debitis error reiciendis tempore doloremque, similique sequi, numquam nesciunt consequatur. Id, perspiciatis. Error, eos corrupti asperiores magni perferendis ipsam

Será? Podemos notar que ainda ficou apertado demais. Para resolver isso, devemos apenas mirar em outro tamanho de tela. Isto é, outro tamanho de tela **a partir do qual** fique com o layout *meio a meio*. Faremos isso mirando em telas médias e acrescentando o texto `md`:

```
<h1 class="mb-3">
  Reviews
</h1>
<div class="row">
  <div class="media col-md-6">
    ...
  </div>
  <div class="media col-md-6">
    ...
  </div>
  <div class="media col-md-6">
```

```
    ...
  </div>
  <div class="media col-md-6">
    ...
  </div>
</div>
```

Agora, tanto a tela do celular quanto a do desktop estarão certas!

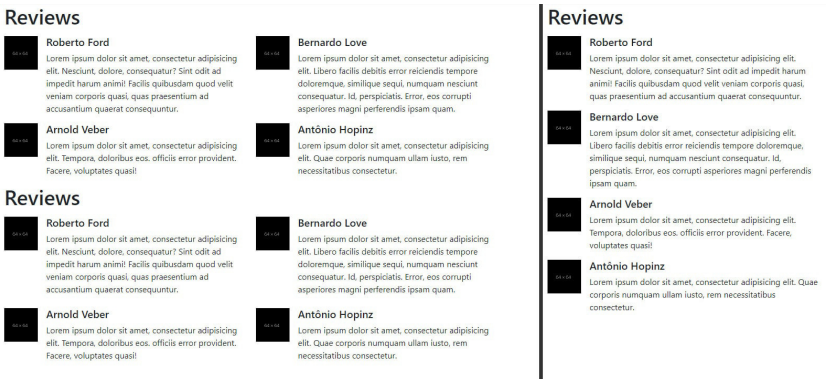

Reviews
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.
Bernardo Love
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero facilis debitis error reiciendis tempore doloremque, similique sequi, numquam nesciunt consequatur. Id, perspiciatis. Error, eos corrupti asperiores magni perferendis ipsam quam.
Arnold Veber
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempora, doloribus eos. officiis error provident. Facere, voluptates quasi!
Antônio Hopinz
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quae corporis numquam ullam iusto, rem necessitatibus consectetur.
Reviews
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.
Bernardo Love
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero facilis debitis error reiciendis tempore doloremque, similique sequi, numquam nesciunt consequatur. Id, perspiciatis. Error, eos corrupti asperiores magni perferendis ipsam quam.
Arnold Veber
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempora, doloribus eos. officiis error provident. Facere, voluptates quasi!
Antônio Hopinz
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quae corporis numquam ullam iusto, rem necessitatibus consectetur.
Reviews
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.
Bernardo Love
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero facilis debitis error reiciendis tempore doloremque, similique sequi, numquam nesciunt consequatur. Id, perspiciatis. Error, eos corrupti asperiores magni perferendis ipsam quam.
Arnold Veber
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempora, doloribus eos. officiis error provident. Facere, voluptates quasi!
Antônio Hopinz
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quae corporis numquam ullam iusto, rem necessitatibus consectetur.

Figura 5.11: Reviews ok no desktop e no mobile

Não esqueçamos do último detalhe: o título `Reviews` está muito encostado na seção de cima, a seção *hall*, e podemos ajustar isso da mesma forma que fizemos no capítulo anterior. Vamos aumentar o `padding-top` com uma das classes de *helper* do Bootstrap, a `pt-5`. Esta é uma opção interessante, pois deixamos um padrão visual, uma vez que já utilizamos exatamente esse `padding` na seção anterior.

```
<section class="reviews container pt-5">
  ...
</section
```

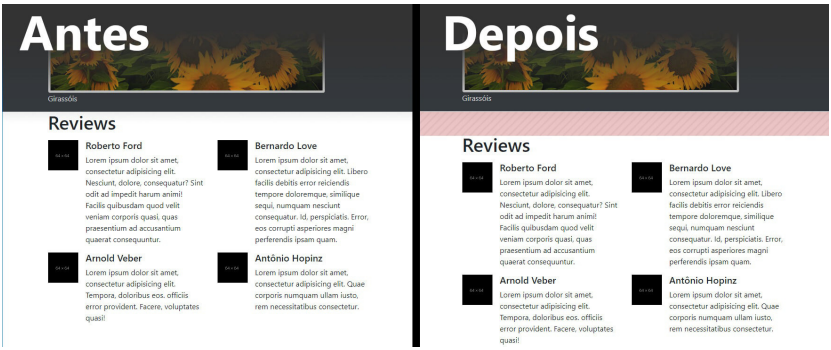

Antes
Girassóis
Reviews
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.
Bernardo Love
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero facilis debitis error reiciendis tempore doloremque, similique sequi, numquam nesciunt consequatur. Id, perspiciatis. Error, eos corrupti asperiores magni perferendis ipsam quam.
Arnold Veber
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempora, doloribus eos, officiis error provident. Facere, voluptates quasi!
Antônio Hopinz
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quae corporis numquam ullam iusto, rem necessitatibus consectetur.
Depois
Girassóis
Reviews
Roberto Ford
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nesciunt, dolore, consequatur? Sint odit ad impedit harum animi! Facilis quibusdam quod velit veniam corporis quasi, quas praesentium ad accusantium quaerat consequuntur.
Bernardo Love
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Libero facilis debitis error reiciendis tempore doloremque, similique sequi, numquam nesciunt consequatur. Id, perspiciatis. Error, eos corrupti asperiores magni perferendis ipsam quam.
Arnold Veber
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Tempora, doloribus eos, officiis error provident. Facere, voluptates quasi!
Antônio Hopinz
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quae corporis numquam ullam iusto, rem necessitatibus consectetur.

Figura 5.12: Comparando o antes e o depois do padding-top

Agora, é só ler o resumo e correr para o próximo capítulo!

5.2 RESUMO

Neste capítulo, fizemos a seção de reviews da Better, e conhecemos o componente *media*, presente desde as versões iniciais do Bootstrap. Além de definir uma `<div>` que representasse cada *review* com a classe `.media`, tivemos de utilizar a `.media-body` para aproveitar os poderosos recursos que o *flexbox* nos proporciona.

Aprendemos também que o Bootstrap possui algumas classes utilitárias focadas em tipografia. Para diminuir um pouco o tamanho da fonte e as margens dos `<h2>`, utilizados nos *reviews*, usamos a classe `h5`.

Vimos que nem sempre o `.container` precisa ficar dentro da seção na qual queremos centralizar o conteúdo. Como no caso da seção trabalhada neste capítulo, não tínhamos nenhuma cor ou imagem de fundo, bastou adicionarmos a classe `.container`

diretamente na `<section>` de reviews. Por fim, lembramos rapidamente de como usar o sistema de *grid* e como ele se comporta quando esquecemos de colocar a `.row`.

No próximo capítulo, veremos um componente novo do Bootstrap 4, presente em muitas interfaces por conta da tendência do *flat design*, o ***card***. Veremos quais são suas características e como ele pode nos ajudar em nosso projeto.

Capítulo 6

CARTAS NA MESA: BRINCANDO COM O CARD

Como citei no capítulo anterior, o Bootstrap tende a acompanhar as tendências do momento, seja com o uso de *flexbox*, do Sass como pré-processador principal, ou até mesmo ao deixar o estilo de seus componentes mais puxados para o *flat design*.

Mais sobre flat design

Recomendo a leitura do post *Afinal, quando posso usar o flat design?* para conhecer mais a respeito: http://blog.alura.com.br/afinal-quando-posso-usar-o-flat-design/.

Um componente interessante – e presente principalmente no *material design* do Google – é o *card*. Literalmente, ele é um elemento que faz referência visual a um cartão físico, presente não só em interfaces *by Google*, como em diversos projetos de UI (*User Interface*).

Ele tenta mesclar um pouco do *flat* com a analogia ao mundo real, bastante útil quando temos informações tabulares, por exemplo. Vamos ver neste capítulo como fazer um *card* usando o Bootstrap e, por fim, terminaremos a primeira e importante etapa do nosso projeto.

Analisando o que será feito

Como sempre, vamos dar uma olhada no que precisamos fazer:

Figura 6.1: Bootstrap em sua segunda versão

Ainda faltam uma seção com os planos da Better e um rodapé, com o que parecem ser ícones de redes sociais. Vamos começar?

6.1 COMEÇANDO COM O PLANO

Primeiramente, vamos focar nos planos. Esta seção será outra centralizada? Podemos criar uma `<section>` com a `.container`, juntamente com outra classe apenas para identificação dela:

```
<section class="container planos">

</section>
```

Vamos colocar o título dessa seção também:

```
<section class="container planos">
  <h1>Planos para todos os bolsos:</h1>
</section>
```

Agora vamos colocar o conteúdo desse primeiro plano, *Silver*, e ir marcando-o de uma forma que faça sentido:

```
<section class="container planos">
  <h1>Planos para todos os bolsos:</h1>
  <h2>Silver</h2>
  <p>19,90</p>
</section>
```

Além do título e do preço, perceba que temos uma lista de vantagens:

```
<section class="container planos">
  <h1>Planos para todos os bolsos:</h1>
  <h2>Silver</h2>
  <p>19,90</p>

  <ul>
    <li>150 fotos</li>
    <li>3 Adesivos</li>
    <li>X camiseta</li>
  </ul>
</section>
```

O usuário precisa do botão para comprar o plano pelo qual tem mais interesse, então, vamos criá-lo:

```
<section class="container planos">

  <h1>Planos para todos os bolsos:</h1>
  <h2>Silver</h2>
  <p>19,90</p>
```

```
  <ul>
    <li>150 fotos</li>
    <li>3 Adesivos</li>
    <li>X camiseta</li>
  </ul>

  <button>
    Quero este!
  </button>

</section>
```

E claro, como teremos mais à frente três planos, são três itens dentro dessa seção. Vamos já separar o plano *silver* dos que faremos ainda neste capítulo:

```
<section class="container planos">
  <h1>Planos para todos os bolsos:</h1>
  <div>
    <h2>Silver</h2>
    <p>19,90</p>

    <ul>
      <li>150 fotos</li>
      <li>3 Adesivos</li>
      <li>X camiseta</li>
    </ul>

    <button>
      Quero este!
    </button>
  </div>
</section>
```

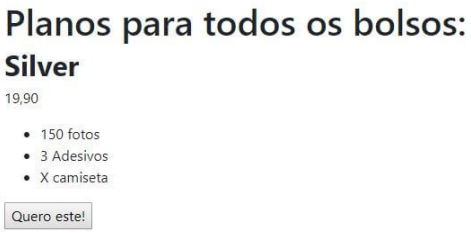


Figura 6.2: Plano sem classe

Poderíamos já duplicar e criar os outros planos, mas vamos deixar isso para quando este primeiro estiver estilizado. Assim, não teremos retrabalho, ok?

6.2 ESTILIZANDO NOSSO PRIMEIRO CARD

No Bootstrap 3, tínhamos disponível o `.panel` em nossa caixa de componentes, que, como o próprio nome diz, casava maravilhosamente bem quando precisávamos colocar algum tipo de painel em nossos projetos. Um exemplo de uso era quando tínhamos seções individuais de um site, ou mesmo o *checkout* em um e-commerce. Enfim, ele era um componente bem flexível.

Pois bem, esse componente evoluiu e tornou-se o já citado *card*. Vamos utilizá-lo aqui da seguinte maneira:

```
<section class="container planos">
  <h1>Planos para todos os bolsos:</h1>
  <div class="card">
    <h2>Silver</h2>
    <p>19,90</p>

    <ul>
      ...
    </ul>
```

```
    <button>
      Quero este!
    </button>
  </div>
</section>
```

Você acha que mudou algo?

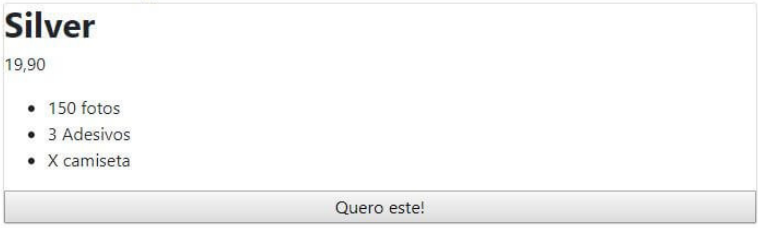


Figura 6.3: Plano sem classe nenhuma

Note que foi colocada uma borda, e o botão agora pega a largura total do *card*, sendo que, na verdade, ele precisava ter um tamanho menor do que tem. Mas não se preocupe, corrigiremos isso em breve.

Vamos estilizar esse título. Segundo o *wireframe*, ele precisa ficar separado do resto do *card*, além de ficar todo em maiúsculo. Podemos fazer isso usando a classe que representa o cabeçalho do *card*, a `card-header` . Além disso, também é possível usar uma daquelas classes de ajuda tipográfica, a `text-uppercase` :

```
<div class="card">
  <h2 class="card-header text-uppercase">Silver</h2>
  <p>19,90</p>

  <ul>
    ...
  </ul>

  <button>
    Quero este!
```

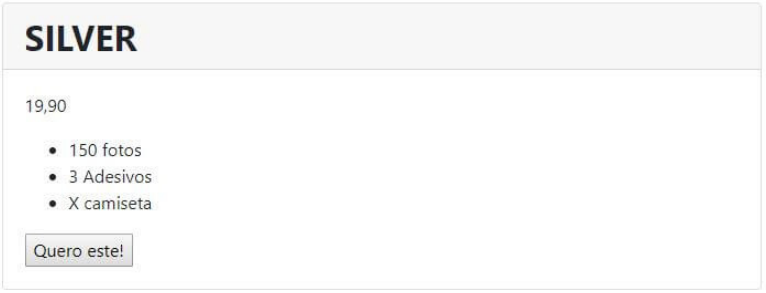
```
  </button>
</div>
```

> **POR QUE SIMPLESMENTE NÃO ESCREVER EM CAIXA ALTA?**
>
> Uma palavra: acessibilidade. Alguns leitores de tela, quando leem uma palavra toda em caixa alta, pensam que se trata de uma sigla. Isto enfraquece a experiência que um cego tem ao navegar em um site.

Se temos um cabeçalho de *card*, será que também temos mais o quê? O corpo e o rodapé! Como não temos nada parecido com um rodapé no *card* do *wireframe*, vamos aplicar somente a classe `.card-body` em volta do conteúdo, abaixo de nosso `<h2>`. Este conteúdo agora está totalmente colado na borda do nosso *card*.

```
<div class="card">
  <h2 class="card-header text-uppercase">Silver</h2>
  <div class="card-body">
    <p>19,90</p>

    <ul>
      ...
    </ul>

    <button>
      Quero este!
    </button>
  </div>
</div>
```

Olhando no browser, veremos:



Mas e este botão cinza e sem vida? Vamos colocar as classes de botão que o Bootstrap nos fornece e que já vimos em capítulos anteriores. A classe `btn` vai deixá-lo com um aspecto melhor e a `btn-primary`, deixá-lo com a cor primária (azul) do nosso projeto. Também podemos deixá-lo um pouco maior com a classe `btn-lg`:

```
<div class="card">
  <h2 class="card-header text-uppercase">Silver</h2>
  <div class="card-body">
    <p>19,90</p>

    <ul>
      ...
    </ul>

    <button class="btn btn-primary btn-lg">
      Quero este!
    </button>
  </div>
</div>
```

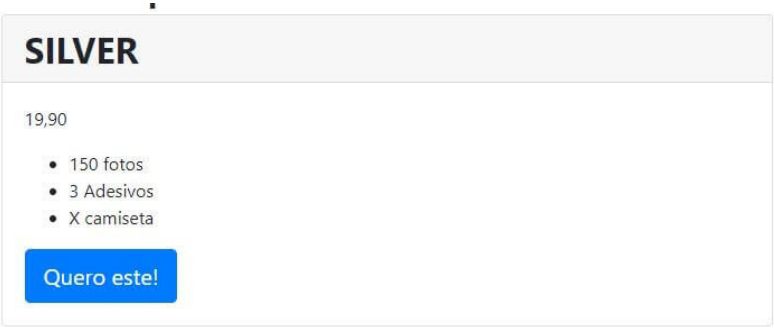


Figura 6.5: Plano com o botão já ok

Estamos quase lá! Não seria melhor se tudo ficasse centralizado, assim como no *wireframe*? Podemos fazer isso em "dois palitos", usando a classe `text-center` direto no *card*:

```
<div class="card text-center">
  <h2 class="card-header text-uppercase">Silver</h2>
  <div class="card-body">
    ...
  </div>
</div>
```

Figura 6.6: Conteúdo do card centralizado

E o que faremos com estes itens e *bullets*?! Não existe isso no

wireframe! Podemos usar o componente de grupo de lista, `.list-group` , a fim de removê-los. Faremos isso juntamente com a classe de cada `<li>` `.list-group-item` , de acordo com a documentação do componente:

```
<div class="card text-center">
  <h2 class="card-header text-uppercase">Silver</h2>
  <div class="card-body">
    <p>19,90</p>

    <ul class="list-group">
      <li class="list-group-item">150 fotos</li>
      <li class="list-group-item">3 Adesivos</li>
      <li class="list-group-item">X camiseta</li>
    </ul>

    <button class="btn btn-primary btn-lg">
      Quero este!
    </button>

  </div>
</div>
```

MAIS POSSIBILIDADES DO GRUPO DE LISTA

Confira outras opções de estilização da `.list-group` : https://getbootstrap.com/docs/4.0/components/list-group/.

Agora, só faltam dois detalhes de design. Perceba que esse grupo de lista cria uma borda em volta de tudo. Podemos tirá-la usando a classe `.list-group-flush` :

```
<div class="card text-center">
  <h2 class="card-header text-uppercase">Silver</h2>
  <div class="card-body">
    <p>19,90</p>
```

```
    <ul class="list-group list-group-flush">
      <li class="list-group-item">150 fotos</li>
      <li class="list-group-item">3 Adesivos</li>
      <li class="list-group-item">X camiseta</li>
    </ul>

    <button class="btn btn-primary btn-lg">
      Quero este!
    </button>

  </div>
</div>
```

Figura 6.7: Card quase ok

Outro detalhe é o botão muito colado ao grupo de lista. Podemos resolver isso dando uma pequena margem, ao colocar a `.mb-2` no grupo de lista:

```
<div class="card text-center">
  <h2 class="card-header text-uppercase">Silver</h2>
  <div class="card-body">
    <p>19,90</p>

    <ul class="list-group list-group-flush mb-2">
      <li class="list-group-item">150 fotos</li>
      <li class="list-group-item">3 Adesivos</li>
```

```
      <li class="list-group-item">X camiseta</li>
    </ul>

    <button class="btn btn-primary btn-lg">
      Quero este!
    </button>

  </div>
</div>
```

Agora podemos ver um *card* em toda sua glória:

Figura 6.8: Card ok

Mas e os outros *cards*?

6.3 APLICANDO A GRID AOS CARDS

Vamos fazer os outros simplesmente copiando e colando todo o *card* duas vezes:

```
<section class="container planos">
  <h1>Planos para todos os bolsos:</h1>

  <div class="card text-center">
    <!-- Conteúdo do plano Silver -->
```

```
  </div> <!-- fim plano silver -->

  <div class="card text-center">
    <!-- Conteúdo do plano Crystal -->
  </div> <!-- fim plano crystal -->

  <div class="card text-center">
    <!-- Conteúdo do plano Gold -->
  </div> <!-- fim plano gold -->
</section>
```

CONTEÚDO REAL

Caso queira, você pode colocar outros valores e outros diferenciais em cada plano. Como é apenas o conteúdo, não impactará em nada nosso trabalho aqui.

Os planos estão um abaixo do outro. Podemos usar o sistema de *grids* para nos ajudar com isso. Cada plano precisa ter 1/3 da largura da seção de planos. Logo, cada um precisará ter quatro colunas.

Não se esqueça de colocar a `.row` em volta de tudo, e que podemos fazer isso com as classes de coluna da seguinte forma:

```
<section class="container planos">
  <h1>Planos para todos os bolsos:</h1>

  <div class="row">

    <div class="col-md-4">
      <div class="card text-center">
        <!-- Conteúdo do plano Silver -->
      </div> <!-- fim plano silver -->
    </div>
```

```
    <div class="col-md-4">
      <div class="card text-center">
        <!-- Conteúdo do plano Crystal -->
      </div> <!-- fim plano crystal -->
    </div>

    <div class="col-md-4">
      <div class="card text-center">
        <!-- Conteúdo do plano Gold -->
      </div> <!-- fim plano gold -->
    </div>

  </div> <!-- fim .row -->
</section>
```

Ao abrirmos o browser, tente reajustar o tamanho da janela e comemore o sucesso!

Figura 6.9: Planos com grid aplicada

6.4 MUDANDO O ESTILO DO PLANO PRINCIPAL

Para induzirmos nosso usuário a se concentrar no plano principal, Crystal, podemos deixá-lo com um aspecto diferente. Chamar mais atenção do usuário é uma prática comum quando falamos tanto de UX quanto de negócio.

Vamos deixar esse plano no topo. Ou seja, o `.card-header` ficará com a cor em destaque usando algumas classes focadas no estilo do Bootstrap. Para isso, podemos usar a cor padrão do framework como cor de fundo em qualquer elemento do nosso site. Vamos usar a classe `bg-primary` :

```
<section class="container planos">

  ...

  <div class="col-md-4">
    <div class="card text-center">
      <h2 class="card-header text-uppercase bg-primary">
        Crystal
      </h2>
      <div class="card-body">
        ...
      </div>
    </div> <!-- fim .card crystal -->
  </div>

  ...

</section>
```

Com isso, o fundo do título fica azul, mas com o texto em preto, deixando o contraste cromático e prejudicado. Podemos mudar sua cor facilmente pela classe `text-light` :

```
<section class="container planos">

  ...

  <div class="col-md-4">
    <div class="card text-center">
      <h2 class="card-header text-uppercase bg-primary text-light
">
        Crystal
      </h2>
      <div class="card-body">
        ...
```

```
      </div>
    </div> <!-- fim .card crystal -->
  </div>

  ...

</section>
```

Ao testarmos no browser, veremos que deu tudo certo! Perceba como fizemos os três componentes rapidamente apenas utilizando meia dúzia de classes. E dessa meia dúzia, algumas já eram velhas conhecidas nossas, como as da *grid* e do botão.

Figura 6.10: Planos ok, com destaque no plano Crystal

6.5 FAZENDO O RODAPÉ

Finalmente chegamos ao *footer* de nossa página. Vamos matar este *job* já começando com uma marcação HTML básica, assim como fizemos durante todo o livro. Primeiro, marcaremos o conteúdo, para depois nos preocuparmos em estilizá-lo, seja com Bootstrap ou não.

```
<footer>
  <span>
    © 2019 Todos os direitos reservados
```

```
  </span>

  <ul>
    <li><a href="http://facebook.com/betterimages">Facebook</a></
li>
    <li><a href="http://twitter.com/betterimages">Twitter</a></li
>
    <li><a href="http://instagram.com/betterimages">Instagram</a>
</li>
  </ul>
</footer>
```

Agora, vamos estilizá-lo! Primeiramente, colocaremos uma cor de fundo em todo o rodapé com a nossa já recém-conhecida `.bg-primary`:

```
<footer class="bg-primary">
  ...
</footer>
```

Figura 6.11: Rodapé totalmente azul

Azul demais, não? Mas, se existe uma cor primária, deve ter uma secundária também! Vamos testar substituindo a classe `.bg-primary` pela `.bg-secondary`:

```
<footer class="bg-secondary">
```

```
  ...
</footer>
```

Agora, com um cinza de fundo, a aparência ficou bem melhor!

> **Outras cores?**
>
> Se você quiser ver as cores prontas que já estão disponíveis no Bootstrap assim como suas respectivas classes, confira este link: https://getbootstrap.com/docs/4.0/utilities/colors/#background-color.

Olhando o *wireframe* do começo deste capítulo, perceba que o fundo desse rodapé, da mesma forma que a seção *hall*, deve encostar nas laterais da janela do browser. E já fizemos isso! Mas como vamos centralizar seu conteúdo? Usando uma `.container` **dentro** do `<footer>`:

```
<footer class="bg-secondary">
  <div class="container">
    <span>
      © 2019 Todos os direitos reservados
    </span>

    <ul>
      ...
    </ul>
  </div>
</footer>
```

Com isso, o conteúdo fica centralizado, e o rodapé fica livre para crescer de acordo com a largura do browser.

Figura 6.12: Rodapé cinza com conteúdo centralizado

Repare que ainda precisamos deixar o `<span>` e `<ul>` lado a lado, seguindo o *wireframe*. Podemos usar o *flexbox* para nos ajudar nisso, com a já conhecida classe `d-flex`. Lembre-se de que o *flexbox* é uma propriedade de exibição que colocamos no elemento-pai para que ele controle seus filhos:

```
<footer class="bg-secondary">
  <div class="container d-flex">
    <span>
      © 2019 Todos os direitos reservados
    </span>

    <ul>
      ...
    </ul>
  </div>
</footer>
```

Isso resolverá nosso problema. Mas como vamos indicar que cada filho dessa `<div>` com a `.d-flex` fica em seu próprio lado? Precisamos que estes elementos fiquem justificados, com um espaço entre eles. Podemos simplesmente aplicar a classe

`.justify-content-between` para chegar ao resultado que queremos:

```
<footer class="bg-secondary">
  <div class="container d-flex justify-content-between">
    <span>
      © 2019 Todos os direitos reservados
    </span>

    <ul>
      ...
    </ul>
  </div>
</footer>
```

Figura 6.13: Rodapé com conteúdo centralizado horizontalmente

Ainda precisamos alinhar verticalmente. *"Ah, agora, será preciso fazer uma classe para isso manualmente, não é, Natan?"*. Não desta vez! O Bootstrap possui todas as propriedades do *flexbox* em formatos de classes! Para alinhar os itens no centro, basta usar a classe `.align-items-center`. Ela inclusive já foi usada anteriormente, no capítulo relacionado ao *hall*:

```
<footer class="bg-secondary">
  <div class="container d-flex justify-content-between align-item
s-center">
    <span>
      © 2019 Todos os direitos reservados
    </span>

    <ul>
      ...
    </ul>
  </div>
</footer>
```

Figura 6.14: Rodapé com conteúdo totalmente centralizado

Calma, logo mais colocaremos os ícones! Assumimos que estas são imagens apenas ilustrativas, de redes sociais e de caráter meramente visual, então, podemos colocá-los via CSS. Mas como poderemos manter o texto ali dentro, por questões de acessibilidade e SEO, ao mesmo tempo em que tiramos sua visualização?

Aqui, vamos utilizar uma técnica muito conhecida no mundo front-end, chamada *image replacement*. Ela serve basicamente para

exibir imagens em algum elemento que originalmente foi feito com texto, escondendo o texto e exibindo a imagem.

Se você já a conhece, deve estar se preparando para dar um `text-indent` negativo. Mas pode ficar tranquilo, pois o Bootstrap tem uma classe para isso, chamada `text-hide`. Vamos aplicá-la aos três links:

```
<ul>
  <li><a href="http://facebook.com/betterimages" class="text-hide
">Facebook</a></li>
  <li><a href="http://twitter.com/betterimages" class="text-hide"
>Twitter</a></li>
  <li><a href="http://instagram.com/betterimages" class="text-hid
e">Instagram</a></li>
</ul>
```

De nada adiantará retirarmos o texto se não colocarmos as imagens. Para isso, teremos de pôr um pouco a mão na massa. O que difere os `<a>` que colocamos ali? O valor do atributo `href` de cada um!

Logo, vamos usar o seletor de atributo do CSS para aplicarmos, como fundo, os logos das redes sociais nesse lugar. Vamos criar uma classe chamada `social` e colocá-la em nossa `<ul>`:

```
<ul class="social">
  <li><a href="http://facebook.com/betterimages" class="text-hide
">Facebook</a></li>
  <li><a href="http://twitter.com/betterimages" class="text-hide"
>Twitter</a></li>
  <li><a href="http://instagram.com/betterimages" class="text-hid
e">Instagram</a></li>
</ul>
```

Agora, no arquivo `better-tema.css`, faremos a brincadeira acontecer ao selecionar os `<a>` pelos seus valores do atributo `href`, especificando cada imagem de fundo e suas dimensões.

```
.social a[href='http://facebook.com/betterimages'] {
  background: url(../img/icone-face.png);
  background-size: 36px 36px
}

.social a[href='http://twitter.com/betterimages'] {
  background: url(../img/icone-twitter.png);
  background-size: 36px 36px
}

.social a[href='http://instagram.com/betterimages'] {
  background: url(../img/icone-insta.png);
  background-size: 36px 36px
}
```

Mas não temos nada, por enquanto. Precisamos permitir que os `<a>` tenham um tamanho, e que as `<li>` fiquem uma ao lado da outra. Isso é possível da seguinte forma:

```
.social a {
  display: block;
  width: 36px;
  height: 36px;
  text-indent: -9999px;

}

.social li {
  display: inline-block;
  margin-left: 1em;
}
```

O último detalhe é que vamos "engordar" um pouco este `<footer>` com as classes de `padding`, apenas para deixar o layout mais agradável, tanto acima quanto embaixo do elemento:

```
<footer class="bg-secondary pt-4 pb-4">
  ...
</footer>
```

Ao testar, temos nosso rodapé terminado:

Figura 6.15: Rodapé ok

Atalho do atalho

O Bootstrap é tão versátil que possui alguns "atalhos" incríveis. Quando queremos colocar o mesmo valor de *padding* no eixo Y – ou seja, tanto para cima quanto para baixo –, poderíamos substituir as classes `pt-4` e `pb-4` pela classe `py-4`. Teste e veja como isso funciona da mesma maneira!

Aproveitando essa "época de engorda", vamos aplicar as mesmas classes nas seções anteriores de planos e reviews, já que seus respectivos conteúdos estão muito grudados aos limites de suas seções:

```
<section class="reviews container pt-5">
  ...
</section>

<section class="container planos pt-4 pb-4">
  ...
```

```
</section>

<footer class="bg-secondary pt-4 pb-4">
  ...
</footer>
```

Com isso, terminamos nosso projeto. Será? Veja o site no desktop e no celular:

6.6 RESUMO

Neste capítulo, com sentimento de despedida, mexemos no *card*, um componente do Bootstrap que evoluiu com o tempo e ficou mais versátil e condizente com as tendências de design atuais. Vimos que ele possui vários pedaços, como cabeçalho e corpo, que manipulamos pelas classes `card-header` e `card-body`.

Brincamos de mexer no CSS, mas sem realmente mexer no CSS, com algumas classes utilitárias, como a `text-uppercase`, `pb-4` e `bg-secondary`. Esta última foi ao mirarmos na cor secundária do framework. Vimos também como deixar uma lista

muito mais apresentável com as classes `list-group` e `list-group-item`.

Fizemos todo o layout usando novamente a *grid*, e as classes que trazem todo o poder e a versatilidade que o *flexbox* nos proporciona com `d-flex` e `justify-content-between`. E para matar o rodapé, fizemos uso da técnica conhecida como *image replacement*, substituindo uma imagem de caráter visual por sua descrição em texto, assim, deixando o rodapé mais acessível.

Espero que você tenha gostado do livro até aqui, pois ainda temos muito o que conversar sobre o Bootstrap.

Capítulo 7

CLASSES SEMÂNTICAS E CSS MAIS LEVE

Como dito nos capítulos iniciais deste livro, muita gente não aprova o uso do Bootstrap por N razões, e uma delas é o uso demasiado de classes e de suas nomenclaturas. Sem contar também a quantidade de componentes não usados que acabamos fazendo o usuário baixar desnecessariamente.

Quantas classes usamos na Better? Umas 70, talvez? E as outras 500 que o Bootstrap possui? Pense naquele usuário que tem 3G (3G só no nome, pois a realidade é "meio G") e está desesperado tentando achar o telefone de um restaurante na rua; fazê-lo baixar o Bootstrap todo não é um pouco de sacanagem da nossa parte?

CSS pesado e classes não semânticas são, dependendo do ponto de vista, uma realidade. Vamos ver agora como mudar essa realidade para acalmar seu amigo que não curte o roxinho Bootstrap.

7.1 DIMINUINDO A POLUIÇÃO NO HTML USANDO SASS

Um desabafo que já ouvi de muitas pessoas que acabaram de

conhecer o Bootstrap é que ele é bacana, mas deixa o HTML extremamente sujo, com muitas classes para um único elemento. E isso ocorre, de fato. Repare no primeiro elemento que criamos, o `<nav>`, ali mesmo na `index.html`:

```
<nav class="navbar navbar-expand-sm navbar-light bg-light">
    ...
</nav>
```

Quatro classes para **um** elemento. Desnecessário? Talvez!

Se não estivéssemos usando o Bootstrap, como faríamos a estilização desse `<nav>`? Colocar as várias declarações CSS de todas essas classes em uma única classe seria uma forma interessante, algo como o código seguinte:

```
.meu-menu {
    position: relative;
    display: flex;
    flex-wrap: wrap;
  ...
  background-color: #f8f9fa !important;
    ...
}
```

Já no HTML, poderíamos substituir todas aquelas quatro classes pela `.meu-menu`, e correr para o abraço com nosso código mais "limpinho". O ponto agora é juntar as quatro classes em uma nova classe batizada por nós.

Para fazer isso, atualmente, precisamos recorrer a um recurso interessante chamado **pré-processadores CSS**. Existem diversos no mercado, como o Less e o Stylus, porém, o Sass é de longe o mais usado. Além disso, ele é o pré-processador oficial em que foi feita a versão 4 do Bootstrap.

Para fazermos isso, vamos precisar entender o que é,

basicamente, um pré-processador e instalar algumas ferramentas. Um pré-processador é uma forma de dar mais poder ao seu CSS, e poder usar variáveis, reaproveitamento de código, aninhamento de regras, funções etc. Mas o browser não entende a extensão do Sass, `.scss`, então vamos ter de compilar qualquer arquivo desse tipo para `.css`:

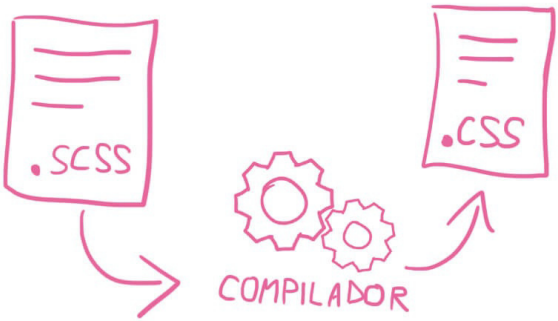


Figura 7.1: SCSS para CSS

7.2 PREPARANDO A INFRA

Para conseguir usar os arquivos-fontes do Bootstrap, precisaremos instalar o NodeJS (https://nodejs.org), o Ruby e o Sass em nossa máquina. Esses arquivos são basicamente o Bootstrap em vários pequenos pedaços. Baixe-o no link, instale-o e faça um teste dando o comando `node -v` no seu prompt/terminal para ver se está tudo bem. Espera-se que apareça a versão que acabou de ser instalada.

Agora vamos instalar o Bootstrap. Dentro da pasta do projeto, digite `cmd .` na barra de endereços da pasta para abrir o seu terminal direto nesse diretório. Se estiver usando Mac ou Linux, apenas navegue via terminal até a raiz do projeto.

No prompt de comando/terminal, vamos instalar os fontes do Bootstrap, dando o seguinte comando:

```
npm install bootstrap@4
```

> O `@` é para indicar qual versão queremos instalar. Se você quiser baixar a última, apenas remova-o deixando somente `bootstrap` no final do comando.

Com isso, o Node baixará o fonte do Bootstrap direto no diretório em que estamos trabalhando. Por enquanto, a estrutura de nossos arquivos está como:

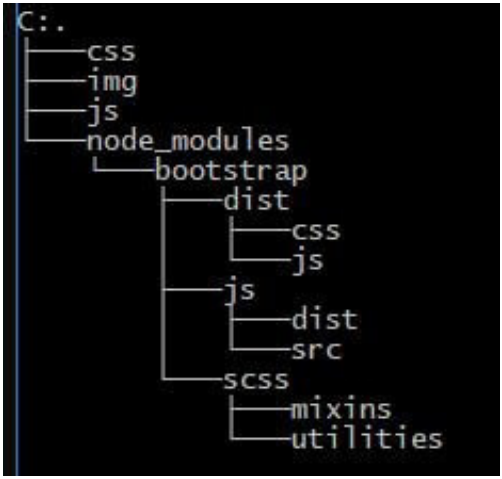


Figura 7.2: Comando tree no prompt mostrando as pastas

Agora, para ajudar a manter o foco do livro no Bootstrap, para instalar propriamente o Ruby e o Sass, siga este guia de instalação: http://blog.caelum.com.br/tutorial-instalando-usando-sass-no-

windows/. É importante também fazer o pequeno teste no final do tutorial.

Com nossa infra pronta, vamos começar a codar!

7.3 JUNTANDO REGRAS CSS EM UMA SÓ CLASSE

Antes de mais nada, qual será arquivo do Bootstrap que deveremos chamar: o que baixamos no começo do projeto ou o módulo que baixamos via NPM? Aquele do começo não está componentizado, logo, precisamos usar este último a fim de deixar futuras manutenções mais fáceis.

Para isso, em `index.html`, vamos **substituir** a chamada do nosso Bootstrap para esse mais novo, agora um módulo Node. No `<head>`, faremos:

```
    <link rel="stylesheet" href="../node_modules/bootstrap/dist/c
ss/bootstrap.min.css">
```

Perceba que tudo continua funcionando e que estamos apontando para o arquivo de **distribuição** (`dist`) do projeto, ou seja, o arquivo final que vai para produção. Mas e os JS que o Bootstrap depende?

Precisaremos instalar o jQuery.js que usamos no menu de navegação, desenvolvido no capítulo inicial do livro. Apesar de já termos esse *script* em nossa pasta `JS`, vamos instalar via node apenas para ficar mais dinâmico de se trabalhar. Então, instalaremos o jQuery, dando o seguinte comando no terminal, na raiz do projeto:

```
npm install jquery@3.3.1
```

Com ele instalado, vamos fazer o mesmo procedimento que fizemos com o CSS do Bootstrap: substituir suas chamadas pelos arquivos de módulo. No final, teremos apenas essas duas chamadas de *scripts*:

```
    <script src="node_modules/jquery/dist/jquery.min.js"></script
>
    <script src="node_modules/bootstrap/dist/js/bootstrap.bundle.
js"></script>

  </body>
</html>
```

Poderíamos sair editando os fontes do Bootstrap, contudo, como vimos nos capítulos anteriores, fazer isso é considerado uma má prática. Se no futuro precisássemos atualizar o framework, teríamos de refazer todas as nossas customizações novamente. Da mesma forma que criamos o arquivo de tema da Better, faremos algo semelhante agora.

Vamos criar agora nosso primeiro arquivo Sass, centralizando todos os arquivos deste tipo no diretório `node_modules/bootstrap/scss`. Lá criaremos um arquivo chamado `better` com a extensão `scss`, ficando então `better.scss`:

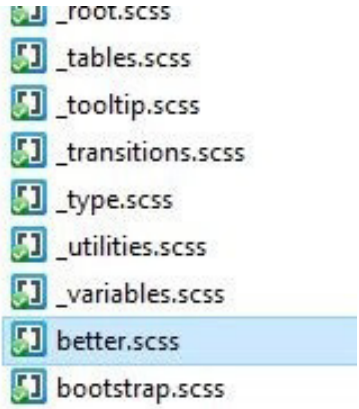


Figura 7.3: Estrutura de pasta mostrando o recém-criado better.scss

É neste arquivo que vamos falar o que queremos exatamente do Bootstrap, podendo ser ele inteiro ou em partes. Agora, de curiosidade, abra o arquivo `/bootstrap/scss/bootstrap.scss`.

Perceba que é nesse arquivo que estão todos os comandos de incluir cada parte do framework! Se não fôssemos usar o CSS responsável pela impressão, por exemplo, poderíamos tirar o *import* do arquivo `print`; isso só seria o começo da nossa limpeza! Basicamente, serão vários *imports* que vamos colocar no `better.scss`.

Vamos abrir o recém-criado `better.scss` no editor e importar o SCSS responsável pelo `navbar`:

```
@import "navbar";
```

Vamos fazer um pequeno teste, da mesma forma do post do link anterior. Como sabemos, os browsers atualmente não entendem arquivos de pré-processadores, logo, precisamos compilar esse `better.scss` em um `better.css` para que o

browser possa entendê-lo. Navegue até a pasta `scss` e entre no terminal usando o truque do `cmd .`, na barra de endereços da pasta.

Agora, compilaremos esse arquivo `better.scss` em `better.css`, com o seguinte comando no terminal:

```
sass --watch better.scss:../dist/css/better.css
```

Com isso, sempre que tiver alguma atualização no `better.scss`, o Sass gerará automaticamente nosso `better.css` na pasta certinha! Mas, logo depois desse comando, teremos o seguinte erro:

```
error _navbar.scss (Line 24: Undefined variable: "$navbar-padding
-y".)
```

O `navbar.scss` está tentando acessar uma variável que não existe, indefinida. Se você deu uma olhada na pasta `scss`, viu que existe um arquivo justamente só de variáveis! Vamos importá-lo no `better.scss` da mesma maneira que fizemos com o `navbar`:

```
@import "variables";
@import "navbar";
```

Mas, temos um erro ainda!

```
error _variables.scss (Line 152: $color: 'theme-color("primary")'
 is not a color for `darken')
```

Ele não encontrou a cor primária do tema padrão. Para conseguirmos que ele a passe, precisamos importar outro arquivo, aquele que gera vários pedaços do Bootstrap, o `"functions"`. Recomendo que, se você curtir programação, dê uma fuçada nesse arquivo para entender a lógica por trás do Bootstrap:

```
@import "functions";
```

```
@import "variables";
@import "navbar";
```

E sim, novamente outro erro:

```
error _navbar.scss (Line 51: Undefined mixin 'hover-focus'.)
```

Este é fácil! Ele não acha o *mixin* `hover-focus` ! Basta incluirmos o arquivo de *mixins*. Mixins basicamente são estilos que podem ser reutilizados por outras regras CSS.

```
@import "functions";
@import "variables";
@import "mixins";
@import "navbar";
```

Agora sim ele compila o `better.css` . Confira se o arquivo está na pasta `CSS` do Bootstrap e abra-o para ver o que criamos.

```
1  .navbar {
2    position: relative;
3    display: flex;
4    flex-wrap: wrap;
5    align-items: center;
6    justify-content: space-between;
7    padding: 0.5rem 1rem; }
8    .navbar > .container,
9    .navbar > .container-fluid {
10     display: flex;
11     flex-wrap: wrap;
12     align-items: center;
13     justify-content: space-between; }
14
15 .navbar-brand {
16   display: inline-block;
17   padding-top: 0.3125rem;
18   padding-bottom: 0.3125rem;
19   margin-right: 1rem;
20   font-size: 1.25rem;
21   line-height: inherit;
22   white-space: nowrap; }
23   .navbar-brand:hover, .navbar-brand:focus {
24     text-decoration: none; }
```

Figura 7.4: Arquivo better.css já gerado do better.scss

Temos cerca de 300 linhas de CSS, entretanto, repare que há apenas o que precisamos para fazer nossa classe `menu`. Agora, no mesmo arquivo `better.scss`, vamos criá-la. Precisamos colocar tudo o que diz respeito à classe `navbar`, `navbar-expand-sm` etc., e, dentro dela, precisamos **estender** essas classes em uma única regra:

```
.meu-menu {
  @extend .navbar;
  @extend .navbar-expand-sm;
  @extend .navbar-light;
  @extend .bg-light;
}
```

Temos outro erro! *Mas será o benedito?*

Lembra que o menu ainda dependia da classe `bg-light`? Precisamos incluir o arquivo em que ela está, o `"utilities"`:

```
@import "functions";
@import "variables";
@import "mixins";
@import "navbar";
@import "utilities";
```

Agora sim! Então, vamos ao `index.html` para tirar aquele monte de classes do menu. Antes, ele estava:

```
<nav class="navbar navbar-expand-sm navbar-light bg-light">
```

E agora ficará da seguinte forma:

```
<nav class="meu-menu">
```

Testando no browser, o menu fica desconfigurado. Parece que falta um CSS ou algo assim, não é?

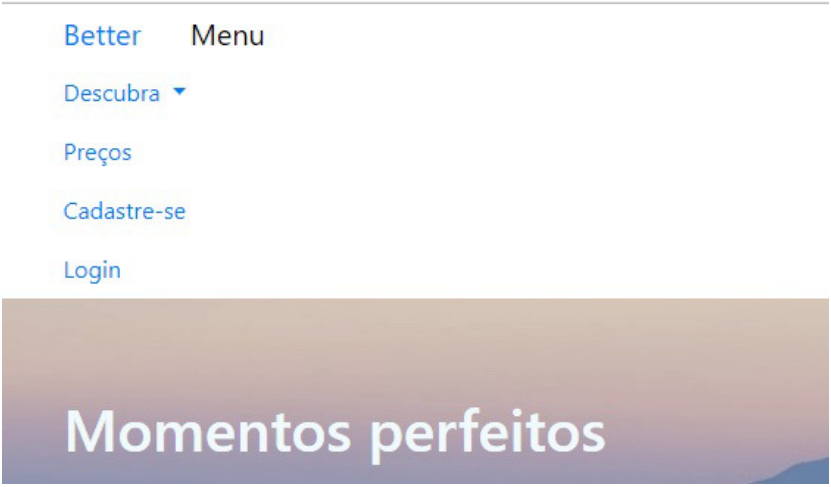


Figura 7.5: Menu desconfigurado

Precisamos chamar, além do Bootstrap e do arquivo de tema que criamos, este recém-criado arquivo com a classe `meu-menu` :

```
  <link rel="stylesheet" href="/node_modules/bootstrap/dist/css/b
ootstrap.css">
  <link rel="stylesheet" href="css/better-tema.css">
  <link rel="stylesheet" href="/node_modules/bootstrap/dist/css/b
etter.css">
```

Testando no browser, tudo ocorre bem! Visualmente está igual, porém, nosso HTML está um "tico" mais limpo.

Aprenda mais sobre os poderes do Sass

Sei que é um momento jabá no meio do livro, mas escrevi um livro sobre Sass que você pode adquirir no site da Casa do Código, em: https://www.casadocodigo.com.br/products/livro-sass.

7.4 ESCOLHENDO O QUE USAR DO BOOTSTRAP

Todo desenvolvedor já ouviu um colega falando: "Ah, mas o Bootstrap é muito pesado!". E, de fato, ele é! Não vou entrar na questão de quão ruim ou irrelevante isso é hoje em dia, mas o ponto é que, das numerosas classes do Bootstrap, tenho certeza de que não estamos utilizando nem metade!

Para tentar enxugar um pouco o tamanho do CSS final que importamos em nosso HTML, podemos importar e gerar um Bootstrap só nosso, só com o que precisamos! Isso vai resultar em um arquivo mais leve para nossos usuários baixarem.

No mesmo arquivo `better.scss`, importaremos outros componentes do Bootstrap além daqueles que já fizemos. Mas quais exatamente serão exportados? Ficar vendo isso manualmente é meio chato; seria mais fácil ir ao `bootstrap.scss`, copiar todos os *imports*, e apenas remover os quais não usamos!

No arquivo `scss/bootstrap.scss`, vamos copiar da linha 8 em diante, e colar tudo isso no `better.scss`, substituindo os

imports que fizemos anteriormente;

Figura 7.6: Vários imports

Se você compilar o CSS disso tudo, gerará um arquivo de 160kb. Meio grande, não?

> **Minificar o arquivo**
>
> Pensando em performance, poderíamos minificar o CSS a fim de deixá-lo bem mais leve. Além de recomendar a leitura do post seguinte, sugiro que procure informações sobre *task runners* front-end, como o Gulp ou o Grunt. Acesse o site e leia: http://blog.caelum.com.br/top-7-praticas-para-um-site-otimizado/.
>
> Se você for usar o Bootstrap baixado direto do site, use a versão minificada do framework, normalmente identificada como `bootstrap.min.css` .

Agora é questão de lembrarmos o que estamos usando e remover o que não estamos, como esse arquivo de `"code"` ou de `"tables"` , por exemplo. Como não estamos usando nenhum trecho de código na Better nem classes de tabelas, podemos removê-los!

Mas e se um dia precisarmos de tabelas? Vamos antecipar isso e apenas comentar esses *imports*:

```
...
@import "images";
//@import "code";
@import "grid";
//@import "tables";
@import "forms";
@import "buttons";
```

...

Também não precisamos do *transitions*, *breadcrumb*, *pagination*, *badge*, *alert*, *progress*, *close* até o *carousel*, e finalmente o *print*.

```
@import "functions";
@import "variables";
@import "mixins";
@import "root";
@import "reboot";
@import "type";
@import "images";
//@import "code";
@import "grid";
//@import "tables";
@import "forms";
@import "buttons";
//@import "transitions";
//@import "dropdown";
@import "button-group";
@import "input-group";
@import "custom-forms";
@import "nav";
@import "navbar";
@import "card";
//@import "breadcrumb";
//@import "pagination";
//@import "badge";
@import "jumbotron";
//@import "alert";
//@import "progress";
@import "media";
//@import "list-group";
//@import "close";
//@import "modal";
//@import "tooltip";
//@import "popover";
//@import "carousel";
@import "utilities";
//@import "print";



.meu-menu {
  @extend .navbar;
  @extend .navbar-expand-sm;
  @extend .navbar-light;
  @extend .bg-light;
}

```

Figura 7.7: Alguns imports

Agora, basta tirar a importação do Bootstrap original do `index.html` e deixar apenas o `better.css`, além do arquivo de

tema:

```
  <link rel="stylesheet" href="/node_modules/bootstrap/dist/css/b
etter.css">
  <link rel="stylesheet" href="css/better-tema.css">
```

Com isso, o `better.css` na prática **será** o Bootstrap, mas feito com apenas o que nosso projeto precisa, deixando seu peso mais leve. E mais um problema do framework que detonamos!

> **LIMPAR MAIS O CSS**
>
> Se você quer passar mais um "pano úmido" no CSS, mas de forma automática, recomendo usar a ferramenta UnCSS. Ela analisa o seu HTML e retira tudo do CSS que não está sendo usado, uma mão na roda! Acesse https://github.com/uncss/uncss.

7.5 RESUMO

Vimos neste capítulo que, mesmo pegando dois grandes problemas do Bootstrap, podemos contorná-los de maneira dinâmica. Primeiro, aprendemos como juntar várias regras CSS em uma única classe, com o objetivo de tentar manter o HTML mais limpo.

Depois, vimos como deixar o Bootstrap apenas com aquilo que precisamos, tornando-o mais leve e personalizado. Agora, vamos para o capítulo final com algumas dicas e mais links úteis!

Capítulo 8

CONSELHOS E DICAS

Durante todo este livro, espero que você tenha aprendido Bootstrap e consiga julgar se vale a pena inseri-lo em seu *workflow*. Espero também que tenha tido curiosidade de fazer seus próprios testes e que tenha curtido o livro.

Meu conselho do começo do livro vale até aqui: conheça bem HTML e CSS, para depois ir para o Bootstrap. Como vimos no projeto Better, hora ou outra precisamos criar algo "na mão", principalmente quando é necessário customizar seu layout criando um tema personalizado.

Mais importante do que usar o Bootstrap é usá-lo com a ciência de que ele é realmente necessário. De nada adianta uma caixa de ferramentas de marceneiro se você precisa furar uma parede. É algo que precisa ser criado muito rápido? Identidade visual não é necessária? Não há um *style guide* da marca da empresa pronto? Não existe um próprio framework da empresa pronto? Então, vá de Bootstrap e seja feliz.

"Poxa Natan, mas meu colega front disse que Bootstrap é uma porcaria!". Infelizmente, ferramentas despertam um sentimento de "defendo o que gosto" mais do que deveriam. Mas, no fim, são apenas ferramentas.

Como vimos no capítulo anterior, sempre podemos dar um jeito em algumas das características do Bootstrap, para que ele fique mais robusto e enxuto. Ou seja, não caia em regras soltas por aí.

Fazer tudo na mão é um orgulho que nós, front-enders, temos (e com razão). Ver um projeto nosso no ar dá esse sentimento, como se fosse um filho. Porém, nosso chefe e/ou cliente não quer saber se usamos Bootstrap, Foundation ou se fizemos tudo manualmente; ele quer o *job* pronto. Claro que também nos deparamos com questões de acessibilidade e semântica, mas, no final das contas, é tudo código; eles querem o resultado, o trabalho concluído, a geração de valor.

O coração do Bootstrap, eu diria, é seu sistema de *grids*, coração este que leva muitos a usar o framework. Se for só para as *grids*, prefira fazer tudo na mão, já que é um código ou um processo a menos que você importará. Sou a favor de fazer tudo na mão, mas quando temos tempo para isso.

8.1 QUAL É O MELHOR FRAMEWORK CSS?

Esta capciosa pergunta é bem polêmica, e os alunos dos cursos presenciais de front-end na Caelum sempre a levantam. Tenho experiência prática com Bootstrap e já brinquei de leve com o Foundation; não acredito em verdades absolutas. Com certeza o Foundation é o mais indicado em determinadas situações ou tipos de projeto, porém, pelo tamanho da comunidade Bootstrap e seus longos anos de casa, eu fico com ele até aparecer algo realmente à altura.

Comunidade Bootstrap

O Bootstrap possui um Slack para acompanharmos discussões e termos contato com outros interessados no framework. Além disso, ainda conseguimos acompanhar seu desenvolvimento, vendo o que está sendo *comitado*. Peça seu convite para entrar no link: https://bootstrap-slack.herokuapp.com/.

Outros pontos do framework

Apesar de não terem sido necessários em nosso projeto que desenvolvemos ao longo do livro, o Bootstrap possui **dezenas** de componentes e guias interessantes que valem a pena serem mencionados aqui. Recomendo uma leitura complementar, da própria documentação (https://getbootstrap.com/), dos seguintes itens:

- Carousel – Imagens que rotacionam, como um site de notícias;
- Accessibility – Dicas para melhorar a acessibilidade do seu projeto;
- Tooltip – Uma pequena janela de ajuda;
- Progress – Barra de progresso;
- Modal – Janela que aparece na frente do conteúdo do site.

8.2 LINKS DE SAIDEIRA

Deixo aqui alguns links que enxergo como boas referências

para discutir e aprender mais sobre o Bootstrap e nossa área de front-end em geral:

Blogs

- http://blog.caelum.com.br – Blog principal do grupo Caelum, que não se limita só a assuntos técnicos, possuindo conteúdos de UX e Agilidade também.
- http://blog.alura.com.br – Blog da plataforma de cursos online Alura, com conteúdos das áreas de Mobile, Dev, Front-end, Infra, Design/UX e Business.
- https://blog.getbootstrap.com/ – Blog da galera que desenvolve o Bootstrap. Sempre quando há novas features ou atualizações, eles postam a respeito.

Temas pagos e free

- BootsWatch – https://bootswatch.com/
- Temas do Bootstrap – https://themes.getbootstrap.com/
- BootstrapMade – https://bootstrapmade.com/

Fóruns

- GitHub Front-End Brasil - https://github.com/frontendbr
- Forúm GUJ - http://www.guj.com.br
- Forúm Tableless - http://forum.tableless.com.br
- Forúm Casa do Código - http://forum.casadocodigo.com.br

Comunidades

- https://bootstrap-slack.herokuapp.com — Slack focado no

Bootstrap, com mais de 20 mil membros.

- https://www.meetup.com/pt-BR/Bootstrap-SP — Meetup em São Paulo. Se não tem um do tema na sua cidade, crie-o! Vamos fomentar discussões e trocar conhecimento em qualquer lugar!

Eventos

- http://conferenciaweb.w3c.br - Conferência W3C;
- https://www.frontinsampa.com.br - Front in Sampa (in Rio, in Fortaleza etc.);
- https://braziljs.org - Maior evento de JavaScript do mundo.

Caso você tenha críticas, sugestões e comentários, pode falar comigo! Estou sempre indo a eventos da área e postando em meu Twitter (*@designernatan*), então, todo feedback é muito bem-vindo. Você também pode entrar em contato pelo fórum da Casa do Código, em http://forum.casadocodigo.com.br.

Meu conselho final abrange mais do que o Bootstrap: nunca pare de estudar. Sempre se informe, pratique e discuta, seja para a ferramenta X, o framework Y ou o processo Z. Vamos perder menos tempo discutindo ferramentas e mais tempo discutindo soluções! Espero, de verdade, que o conteúdo deste livro faça diferença em sua vida, e até a próxima!